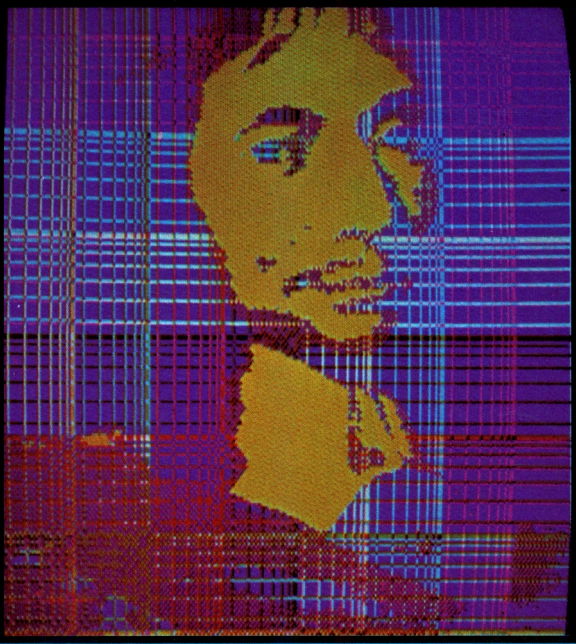
E. FLOEGEL

DU BASIC AU PASCAL

UNE INTRODUCTION AU PASCAL



MICRO SYSTEMES

ETSF

Collection
MICRO-SYSTEMES
dirigée par ALAIN TAILLIAR
rédacteur en chef de la revue MICRO-SYSTEMES

DU BASIC
AU PASCAL

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que « les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants-droit ou ayants-cause, est illicite » (alinéa 1er de l'art. 40). Cette représentation ou reproduction, par quelque procédé que ce soit constituerait donc une contrefaçon sanctionnée par les Art. 425 et suivants du code pénal.

Cet ouvrage est paru en version originale de langue allemande sous le titre « PASCAL Handbuch, von BASIC zu PASCAL », aux Editions Ing. W. Hofacker GmbH Verlag de Munich.
© 1981 by Ing. W. Hofacker, Postfach 75 437, D-8000 MUNICH 75.
© 1983 by Editions Techniques et Scientifiques Françaises, 2-12, rue de Bellevue, 75019 PARIS pour la version de langue française.

ISBN 2-85535-023-9

Photo de couverture : J.-F. COLONNA

E. Floegel

DU BASIC AU PASCAL

Traduit et adapté par Daniel Ranc

Diffusion

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES

2 à 12, rue de Bellevue, 75940 Paris Cedex 19

Sommaire

1. Pourquoi le PASCAL ? 9

2. Les éléments de base du PASCAL 13
   2.1 La construction d'un programme PASCAL 13
   2.2 Les noms 16
      2.2.1 Description des éléments du langage..... 17
   2.3 Représentation des nombres..... 18

3. La section de déclaration 21
   3.1. Déclaration des constantes 21
   3.2. Déclaration des variables..... 22
      3.2.1 Déclaration des tableaux de nombres et de caractères 22
   3.3 Affectation des variables..... 23
   3.4 Procédures d'entrée-sortie 25
      3.4.1 L'instruction WRITE 26
      3.4.2 Mise au format des affichages..... 26
      3.4.3 L'instruction READ..... 28
   3.5 Fonctions standard 29

4. Les boucles et les imbrications..... 31
   4.1 L'instruction FOR 31
   4.2 L'instruction REPEAT..... 35
   4.3 L'instruction WHILE 36
   4.4 L'instruction conditionnelle IF... THEN...ELSE..... 39
   4.5 L'instruction CASE...OF 42

5. Données et ensembles de type particulier..... 45
   5.1 Définition du type des tableaux 45
   5.2 Le type RECORD 48
      5.2.1 L'instruction WITH 49
   5.3 Les ensembles et leurs dénombrements..... 49

6. Procédures et fonctions 53
6.1 Procédures 53
6.2 Les fonctions 60
6.3 Procédures et fonctions récursives 61

7. Structures de données dynamiques 65
7.1 Construction d'une liste 66
7.2 Insertion et suppression au sein de listes classées 69
7.3 Fichiers séquentiels 69

8. Intégration de programmes machine aux programmes en PASCAL 79
8.1 Un exemple : le traitement de données résultant d'une mesure 79

9. Une collection de différents programmes 85
9.2 Méthode de HORNER 88
9.3 Multiplication des polynômes 88
9.4 Addition de polynômes 90
9.5 Dérivée d'un polynôme 92
9.6 Analyse de Fourier 92
9.7 Comparaison entre deux lettres 94
9.8 Programme SPLIT 94
9.9 Programme QUICK 95

10. Comment élaborer un programme en PASCAL 99
10.1 Exemple de programmation sur APPLE II 99
10.2 Quelques particularités de l'Apple 101
10.2.1 Affichage de résultats par l'imprimante . 101
10.2.2 La procédure SEEK 101
10.2.3 La procédure PAGE 102
10.2.4 La procédure GOTOXY 102

11. PASCAL UCSD sur le C4PMF d'Ohio Scientific 105

Introduction

La maîtrise d'au moins un langage de programmation est, aujourd'hui, une nécessité. Un langage dont la nature semble, de prime abord, sans importance. L'expérience montre cependant que le PASCAL, par sa construction logique, permet non seulement un apprentissage facile, mais aussi incite le programmeur à écrire des programmes clairs. Le présent ouvrage s'efforce d'une part de permettre l'accès au PASCAL et, d'autre part, à l'intention de tous ceux qui, jusqu'à présent, n'utilisèrent que le BASIC, de faciliter la reconversion au PASCAL. Quelques mots suffisent pour écrire un programme prêt à fonctionner. Par contre, plus les applications deviendront complexes, plus il faudra posséder à fond le langage.

Ces constatations inspirèrent la présentation du livre. Au début ne seront utilisées que des structures de données simples, et, plus loin, apparaîtront les procédures, les fonctions et les listes.

Les programmes donnés en exemple ont pour objet de montrer l'essentiel, ils sont donc conçus de manière simple. Les premiers programmes sont même accompagnés de leur équivalent en BASIC.

Mais tout programme est écrit dans le but de l'exécuter. Pour cela, il faut un calculateur et un système, dont l'apprentissage est souvent plus difficile que celui du langage lui-même. Pour cette raison, ce livre contient un chapitre décrivant la mise en œuvre d'un programme sur un ordinateur (Apple II).

Monsieur Franz Ende a, très aimablement, ajouté un chapitre sur le PASCAL et l'OHIO C4P.

Holkirchen, printemps 1981.

E.Floegel

1

Pourquoi le PASCAL ?

L'ère des machines à calculer débuta avec l'idée de stocker les données et les programmes sous forme d'une suite de bits dans une même mémoire. L'introduction de ces bits était lente et laborieuse. Pour cette raison, l'on a conçu les langages d'assemblage, qui transforment automatiquement le programme en code machine et utilisent une écriture facile à apprendre et à retenir. La formulation de fonctions mathématiques posait malgré tout d'énormes problèmes. On a alors développé le premier langage de programmation « évolué » : le **FOR**mula **TRAN**slator ou **FORTRAN.** Le succès de ce langage reposait sur l'emploi de mots simplifiant et réduisant le temps de programmation. Malgré cela, FORTRAN ne put répondre à tous les besoins. L'absence de sous-programmes récursifs, c'est-à-dire de sous-programmes pouvant se rappeler eux-mêmes, posait des difficultés à la résolution de nombreux problèmes mathématiques.

Un nouveau langage, l'**ALGO**rithmic **L**anguage, ou **ALGOL** en abrégé, devait alors satisfaire aux nouvelles conditions. L'implémentation de ce langage fut difficile et ne fut réalisée que dans une version simplifiée, l'ALGOL 60. L'ALGOL comportait d'importantes améliorations par rapport au FORTRAN, mais son emploi ne dépassa pas, malgré tout, le domaine des universités. Les deux langages donnent le meilleur d'eux-mêmes dans la description des modèles mathématiques. Parallèlement apparu le **CO**mmon **B**usiness **O**riented **L**anguage **(COBOL),** pour la description des problèmes commerciaux.

Au cours des années soixante furent créés d'autres langages, lesquels étaient construits spécialement pour certains problèmes, comme par exemple **LISP,** un langage d'intelligence artificielle, ou bien **APL** (**A** Programming **L**anguage), **SIMULA** et bien d'autres. La figure 1.1 en donne le résumé chronologique.

FORTRAN	1957
ALGOL	1960
LISP	1961
BASIC	1965
PL/1	1965
APL	1967
PASCAL	1971

Fig. 1.1. – Récapitulatif des dates d'apparition des langages de programmation.

Vers la fin des années soixante, le professeur N. Wirth recherchait un langage pouvant servir non seulement à l'apprentissage de la programmation, mais aussi à la description des algorithmes et à une systématisation de la construction des programmes. Comme aucun des langages existants ne correspondait à ce concept, il entreprit le développement d'un nouveau langage basé sur l'ALGOL et lui donna le nom du mathématicien et constructeur des premières machines à calculer : Pascal (1623-1662).

Par rapport à l'ALGOL, l'extension réside principalement dans l'introduction de nouvelles structures de données. Le **PASCAL** offre ainsi la possibilité de définir un tableau de données comprenant non seulement des nombres, mais aussi des chaînes de caractères. Un exemple en est un fichier-clients qui contiendrait à la fois les numéros des clients, le nom des firmes, leurs adresses et leurs numéros de téléphone. Tout ceci peut être consigné dans un seul tableau de données (RECORD).

Mais le PASCAL s'utilise non seulement en programmation, mais aussi pour la formulation des algorithmes (ce sont des procédés et des modèles mathématiques de résolution des problèmes complexes). C'est ainsi qu'apparut le concept de la programmation structurée, qui, d'ailleurs, ne fut jamais défini de façon précise, mais dont une bonne traduction pourrait être « un procédé pour l'écriture de programme clairs et lisibles ».

Un autre langage de programmation, nommé **B**eginners **A**ll purpose **S**ymbolic **I**nstruction **C**ode, **BASIC** en abrégé, vit le jour au début des années soixante. Ce langage interprété est composé d'instructions très simples, d'apprentissage rapide. Autre avantage : seule une connaissance minimale du système est nécessaire. Cela facilite l'usage du matériel et du logiciel.

Le BASIC est donc devenu le langage le plus répandu des

micro-ordinateurs et des ordinateurs individuels ; avec lui, de nombreux problèmes se laissent programmer rapidement et facilement, pour être résolus par le calculateur.

Mais pourquoi, alors, le PASCAL ?

Aujourd'hui, l'on exige d'un programme certaines caractéristiques que le BASIC ne peut satisfaire. Ce sont par exemple :

a) La vérification aisée de parties de programme. Une partie de programme est facilement vérifiable, en ce qui concerne son exactitude et sa clarté, lorsqu'elle possède une entrée et une sortie. Et c'est justement ce qui, en BASIC, est empêché par l'instruction GOTO (d'où une programmation en « spaghettis »). En PASCAL, GOTO existe, mais est indésirable.

b) La clarté des programmes. L'utilisation de noms de programmes caractéristiques (identificateurs), et leur organisation, les rend plus lisibles. Mais ceci relève plutôt des habitudes du programmeur. Il est toujours possible d'écrire des programmes PASCAL illisibles. Mais, ici, un programme lisible ne nécessite pas une documentation longue de plusieurs pages.

c) La subdivision des programmes longs en sections munies de conditions de transfert définies et la détermination de variables globales et locales. Ceci permet à plusieurs programmeurs de travailler sur les différentes parties d'un programme, simultanément.

D'un autre côté, le PASCAL a aussi ses défauts. D'une part, il manque la notion de définition dynamique de tableau, et la précision de calcul est limitée. D'autre part, l'emploi du langage PASCAL dépend du confort du système. Insistons bien sur le fait que le PASCAL et le système d'exploitation sont deux choses distinctes. Le PASCAL est identique sur tous les calculateurs, contrairement aux systèmes d'exploitation.

Ces réflexions mènent à la conséquence suivante :

En ce qui concerne les programmes courts, le BASIC reste la solution la plus simple et la plus rapide.

Les programmes longs et compliqués devraient être écrits en PASCAL. Une meilleure lisibilité, la documentation et une vérification facilitée sont ici des arguments décisifs.

Pour faciliter l'accès au PASCAL à tous ceux qui, jusqu'à maintenant, ne connaissaient que le BASIC, nous présenterons, au début et partout où cela sera possible, les programmes BASIC et PASCAL face à face.

2

Les éléments de base du PASCAL

2.1 La construction d'un programme PASCAL

Tout programme PASCAL est construit suivant un schéma bien défini. Ce type de schéma nous est connu dans la vie quotidienne sous la forme de mode d'emploi, d'instructions de montage ou de recette de cuisine. Examinons donc les instructions de montage (fig. 2.1).

La construction des programmes PASCAL est tout à fait identique à ce schéma. Trois parties composent le programme : l'en-tête du programme, la section de déclaration et la section d'exécution ou programme principal.

La figure 2.1 montre une comparaison entre des instructions de montage et le programme PASCAL nommé EXEMPLE.

PROGRAMME EXEMPLE

Ceci est la tête de programme du programme nommé EXEMPLE. Le signe ; qui suit conclut cette instruction. Ce caractère est toujours employé, en PASCAL, pour séparer les instructions. Sur certains compilateurs, le nom du programme doit être suivi de la déclaration des fichiers employés, surtout les fichiers d'entrée et de sortie. Ces instructions seront données entre parenthèses.

PROGRAMME EXEMPLE (INPUT, OUTPUT)

La section de déclaration suit la tête de programme. Ici seront définies toutes les constantes, variables, procédures et fonctions utilisées par la section d'exécution. La section d'exécution commence par un BEGIN et se termine par un END.. Le point qui suit END indique la fin du programme EXEMPLE.

L'étoile	Désignation de l'objet	
Elle se compose de seize brins de paille et d'herbe, de longueur et d'épaisseur identiques (20 à 22 cm), à attacher de la manière suivante :	Description des composants nécessaires.	
1er temps : mettre deux brins en croix 2e temps : insérer deux brins d'herbe dans les espaces 3e temps : mettre deux brins en croix 4e temps : insérer deux brins d'herbe dans les espaces.	Quelles sont les opérations à effectuer ?	
Ceci donne la première étoile. Nous avons besoin, ici, de quatre de ces étoiles, lesquelles seront ensuite reliées pour former deux étoiles, celles-ci formant à leur tour l'étoile finale. Lors de la réunion et de la superposition des différentes étoiles, il faut veiller à ce que les brins de paille et d'herbe soient en contact entre eux, respectivement. Toutes les autres étoiles sont formées à partir de brins de paille entiers. Avant de les manipuler, il faut les tremper dans de l'eau chaude. Un fil solide est nécessaire, les étoiles étant plus lourdes.	Comment l'objet est-il construit ?	

Fig. 2. 1. – Comparaison entre un mode d'emploi et un programme PASCAL.

```
PROGRAM EXEMPLE;

CONST N=100;

VAR I,L,M:INTEGER;
    LISTE:ARRAY[1..N] OF INTEGER;

PROCEDURE TRI;
VAR B,I,J:INTEGER;

BEGIN
  J:=1;
  REPEAT
    I:=J+1;
    B:=LISTE[J+1];
    REPEAT
      I:=I-1;
      IF B<LISTE[I] THEN LISTE[I+1] :=LISTE[I]
                    ELSE LISTE[I+1] :=B;
   UNTIL((I=1) OR (B=LISTE[I+1]));
   IF I=1 THEN LISTE[1]:=B;
   J:=J+1;
  UNTIL J=N;
END;   (*TRI*)

BEGIN
  FOR I:=1 TO N DO LISTE[I]:=N+1-I;
  TRI;
  FOR I:=1 TO N DO WRITELN(LISTE[I]);
END
```

Comme nous le verrons sur les exemples suivants, le PASCAL est un langage très axé sur l'emploi de procédures. Dans la plupart des cas, cela conduit à ce que la section de déclaration prenne la majeure partie de la place, cependant que la section d'exécution se réduit à l'appel de procédures.

2.2 Les noms

Les constantes, les variables, les procédures, etc., sont caractérisées par des noms. Leur longueur est indifférente, mais ils doivent commencer par une lettre ; ils peuvent contenir des nombres, mais pas de signes de ponctuation.

Exemple :

DEBUT, NOMCOURT, CLASSE5B, CL5B sont des noms autorisés, alors que les suivants ne le sont pas : CLASSE 5B, 3EMEMENT, QUATRE. CINQ.

De la même façon, il est impossible d'utiliser les mots-instructions PASCAL. Ces noms protégés sont reproduits à la figure 2.2.

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNTO	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

Fig. 2.2. – Noms protégés en PASCAL.

Les noms sont différenciés sur leurs huit premières lettres. Ainsi, le compilateur considérera que FICHIERS et FICHIERSADHERENTS sont identiques. La lisibilité des programmes dépend en grande partie d'un bon choix des dénominations. Ainsi, des noms comme NOTE ou RESULTAT seront immédiatement compréhensibles au lecteur, alors que des expressions telles que NN ou ER ne peuvent souvent pas être identifiées.

2.2.1 *Description des éléments du langage*

Deux types de descriptions seront utilisés pour décrire les éléments du PASCAL. D'une part, une variante de la FORME BACKUS-NAUR (FBN), dans laquelle les commentaires sont écrits entre crochets, <>, les répétitions entre accolades, {}, et une alternative par un trait vertical, |.

La combinaison ::= signifie « correspond à » et sert d'attribut à des symboles explicatifs.

Rappelons que ces caractères, nommés METASYMBOLES, ne sont pas des éléments du langage PASCAL et ne servent qu'à la description.

La définition d'un nom, en PASCAL, avec ces conventions d'écriture, ressemblerait à ceci :

<NOM> ::= <LETTRE> {<LETTRE> | <CHIFFRE> }

Traduit en français, cela donnerait :

Le NOM dans un programme CORRESPOND A une LETTRE, suivie de LETTRES ou de CHIFFRES.

Une autre façon de décrire les programmes est le diagramme syntaxique. Cette description graphique traduit les terminaux par des rectangles arrondis ou par des cercles, les non-terminaux par des rectangles simples (fig. 2.3). Les terminaux sont ici les éléments ne pouvant plus être dérivés.

Terminaux Non-terminaux

Fig. 2.3. – Les éléments du diagramme syntaxique.

Ce sont soit des instructions du PASCAL, soit des mots du langage commun, ne nécessitant plus d'explication particulière. Les non-terminaux, par contre, sont reliés à d'autres diagrammes syntaxiques les décrivant. La figure 2.4 représente le diagramme d'un nom. Les mots LETTRE et CHIFFRE ne nécessitent pas d'autres explications, ce sont des terminaux. Les symboles sont reliés par des traits, les flèches indiquent un sens de fonctionnement possible.

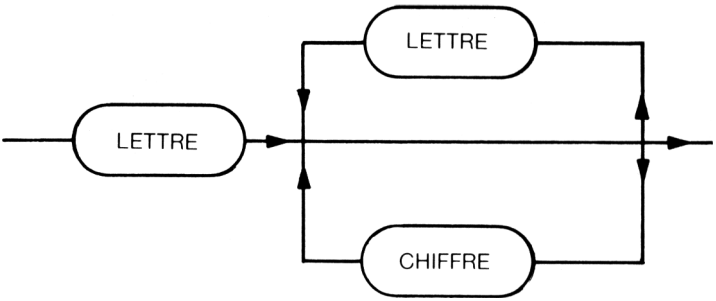


Fig. 2.4. – Diagramme syntaxique NOM.

2.3 Représentation des nombres

Deux sortes de nombres peuvent exister dans un programme. Ce sont les nombres entiers (INTEGER) et les nombres réels (REAL). L'écriture des entiers est univoque.

<NOMBRE ENTIER> ::= <SIGNE> | <> <CHIFFRE>
{ <CHIFFRE> }
<SIGNE> ::= < +|–>

Exemples :
–512, 386, 1001, etc.

Le domaine de calcul, pour les entiers, s'étend de –32768 à 32767.

Les réels peuvent être écrits soit en nombre simple, soit avec un point décimal, soit encore en notation scientifique. Lorsque l'on emploie le point décimal, il doit y avoir un nombre avant et après celui-ci.

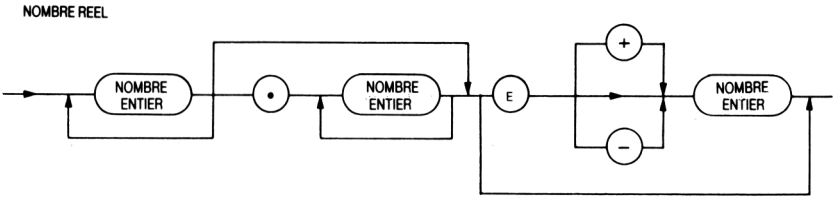
Exemples : 5, 3.14, 10E5, 1E-5, 2.0E-3

Sont interdits, par contre : 5., .31, 1.E-4

<REEL> ::=
<ENTIER> | <ENTIER> . <NOMBRE> |
<ENTIER> . <NOMBRE> <EXPOSANT>
<NOMBRE> ::= <CHIFFRE> <CHIFFRE>
<EXPOSANT> ::=E <ENTIER>

La figure 2.5 montre le diagramme syntaxique des réels.

On trouve parfois, dans la littérature, la notion de « nombre réel » remplacée par l'expression « virgule flottante ».



Fig, 2.5. – Diagramme syntaxique des réels.

3

La section de déclaration

Toutes les constantes, les données, les variables, procédures et fonctions utilisées dans la section d'exécution sont définies par un nom, dans la section de déclaration. Dans le cas des constantes, l'on définit en même temps leur valeur ; les valeurs des variables, par contre, ne seront attribuées que dans le programme principal. Les déclarations se font dans l'ordre suivant :

<SECTION DE DECLARATION> ::= <CONSTANTES>
<TYPES>
<VARIABLES>
<PROCEDURES>
<FONCTIONS>

Il arrive qu'une partie de la section soit vide ; en effet, si, par exemple, un certain programme n'emploie pas de constantes, il n'est pas nécessaire d'en définir.

3.1 déclaration des constantes

L'on déclare les constantes de la manière suivante :

CONST <NOM> = <CONSTANTE> ;
{ <NOM> = <CONSTANTE> ; }

<CONSTANTE> ::= <NOMBRE ENTIER> | <NOMBRE REEL> |
<CHAINE DE CARACTERES> | <TERME>

<CHAINE DE CARACTERES> ::= <CARACTERE>
{<CARACTERE>}

<CARACTERE> ::=' <LETTRE> | <CHIFFRE> | <SYMBOLE>'

<TERME> ::= <CONSTANTE> <OPERATEUR>
<CONSTANTE>

<OPERATEUR> ::= < + | − | * | / >

Ce qui permet de définir :

```
CONST  PI (1) = 3.14159 ; UN= 1 ; VIDE =' ' ;
       PIDEMI = PI/2 ; ETOILES = '******' ;
```

Le fait de définir PI ET PIDEMI dans l'ordre inverse (PIDEMI =PI/2 ; PI = 3.14159) aurait entraîné le compilateur à déclarer une erreur, car, lors de la première définition, PI était inconnu. Les lettres et les caractères peuvent aussi être déclarés en tant que constantes. Celles-ci sont alors placées entre guillemets (').

3.2 Déclaration des variables

Avant de déclarer les variables, il faut déclarer les types de données particuliers. Ceux-ci seront cependant traités à part, dans un chapitre suivant.

Une variable est déclarée par l'attribution d'un nom et du type.

```
VAR < NOM>{, <NOM>}:< TYPE> ;
{<NOM> , <NOM> : <TYPE> ;}
<TYPE> ::= <ENTIER> | <REEL>
| <BOOLEEN> | <CHAINE>
```

Nous ne considérons ici que les types de base. Ce sont INTEGER (entiers), REAL (réels), BOOLEAN (variables booléenes prenant les valeurs TRUE et FALSE) et CHAR (caractères).

Exemple :

```
VAR  COMPTEUR, INDEX : INTEGER ;
     RESULTAT, SOMME : REAL ;
     EST : BOOLEAN ;
     TUT : CHAR ;
```

Les noms ne doivent être employés qu'une seule fois. Il est illégal d'employer le nom COMPTEUR à la fois en INTEGER et en REAL.

3.2.1 Déclaration des tableaux de nombres et de caractères

Les tableaux numériques sont déclarés de la manière suivante :

```
<NOM> : ARRAY [<LIMITES DES INDEX>]
OF <TYPE DES ELEMENTS> ;
<LIMITES DES INDEX> ::= <LI> .. <LS> {,<LI..LS>}
```

(1) PI = π = 3,14159.

Les nombres LI (limite inférieure) et LS (limite supérieure) doivent être des nombres entiers. S'ils sont désignés par un nom de variable, il faut que celui-ci ait été défini au préalable. Ici apparaît une des plus grandes faiblesses du PASCAL : l'impossibilité de définir des tableaux dynamiques.

<TYPE DES ELEMENTS> ::= <ENTIER> | <REEL>
| <LETTRE> | <TYPES PARTICULIERS>

Exemples :

VECTEUR : ARRAY [1..3] OF REAL ;

TABLEAU : ARRAY [0..9] OF INTEGER ;

NOM : ARRAY [1..25] OF CHAR ;

De la même façon, il est possible de définir des tableaux à plusieurs dimensions :

MATRICE : ARRAY [1..3, 1..5] OF REAL ;

Comme les éléments d'un tableau peuvent aussi être des lettres ou des caractères, il est possible de former, de la même façon, des tableaux avec des chaînes de caractères.

Remarque :

Sur quelques systèmes, les chaînes de caractères sont désignées par STRING, TEXT ou ALFA. La version UCSD de STRING comprend jusqu'à 255 lettres ou signes au maximum, si l'on ne définit pas de limite supérieure. Cette limite, le cas échéant, sera indiquée entre crochets. Les expressions suivantes sont donc équivalentes :

NOM : ARRAY [1..25] OF CHAR ;
NOM : STRING [25] ;

Les processus de déclaration des procédures et des fonctions ne seront décrits que dans un chapitre ultérieur. L'écriture de programmes simples nécessite encore deux choses : l'affectation de valeurs aux variables et des instructions d'entrée-sortie.

3.3 Affectation des variables

En PASCAL, l'affectation des variables est réalisée par la combinaison de caractères : = qui, si on la traduisait en langage commun, signifierait « est remplacé par ».

<VARIABLE> : = <EXPRESSION> ;

D'autres langages se contentent du signe « égal ». Mais ceci désoriente souvent le programmeur débutant, surtout lorsqu'apparaissent des affectations de la forme suivante :

NOMBRE = NOMBRE + 1

Expression qui, en apparence, est fausse, puisque la valeur de la variable NOMBRE ne peut être égale à la valeur de NOMBRE augmentée d'un. En PASCAL, par contre, l'on écrirait :

NOMBRE : = NOMBRE + 1

Qui signifierait que la valeur de la variable NOMBRE est remplacée par la valeur de NOMBRE, plus un. Ou encore, tout simplement, que NOMBRE est remplacé par NOMBRE + 1. Examinons maintenant l'affectation des constantes. C'est le signe « égal » qui est utilisé ici, ce qui, formellement, est cohérent puisque PI est bien « égal » à 3.14159...

L'utilisation des opérateurs de calcul +,–,*,/ suppose l'emploi des conventions habituelles : multiplication et division avant addition et soustraction. L'emploi de parenthèses devra assurer l'ordre exact des opérations arithmétiques. L'ensemble des caractères employés en PASCAL est indiqué à la figure 3.1.

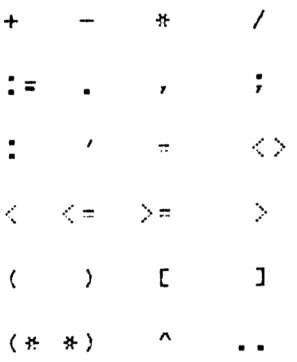


Fig. 3.1. – Caractères utilisés en PASCAL.

Exemples :

SOMME : = SOMME + NOMBRE ;

RESULTAT : = NOMBRE/(2*PI) ;

Toute variable placée à droite d'une affectation devra, au préalable, être définie par une valeur. Dans le cas contraire, les résultats

des calculs seront aléatoires. Cette faute se produit souvent chez les programmeurs ayant, précédemment, travaillé uniquement en BASIC.

Un petit programme de calcul de TVA en donnera une illustration : le programme 3.3.1.

```
PROGRAM  PLUSVALUE;

    CONST TVA=0,186;

    VAR   SOMME,SOMMEFIN,TAXE:REAL;

    BEGIN
      WRITE("SOMME=");READLN(SOMME);
      TAXE:=TVA*SOMME;
      SOMMEFIN:=SOMME+TAXE;
      WRITELN("SOMMEFIN=",SOMMEFIN:7:2);
    END.
```

```
10 REM TVA
20 INPUT "SOMME: ";B
30 S = 0,186*B
40 S = S+B
50 PRINT ("SOMMEFIN: ");S
60 END
```

Programme 3.3.1 : Calcul de la taxe à la valeur ajoutée

3.4 Procédures d'entrée-sortie

Le programme 3.3.1 emploie les procédures standard READ, ou plutôt READLN, et WRITE/WRITELN. Ces procédures contrôlent les échanges de données entre le calculateur et les fichiers. En PASCAL, les périphériques d'entrée-sortie sont également considérés comme des fichiers. Le fichier d'entrée normal étant le clavier, celui de sortie étant l'écran. L'écriture et la lecture de fichiers en mémoire de masse (disque floppy) seront évoquées plus tard.

3.4.1 L'instruction WRITE

L'instruction WRITE prend l'aspect suivant :

WRITE (<VARIABLE> {,< VARIABLE>}) ;

<VARIABLE> ::= <ENTIER> | <REEL> | <CARACTERE>
| <CHAINE DE CARACTERES>

Exemple :

WRITE ('RESULTAT = ',RESULTAT) ;

Cette instruction emploie deux variables : la chaîne de caractères 'RESULTAT', et la valeur numérique que prend la variable RESULTAT à cet instant précis.

Lorsqu'une première instruction WRITE est suivie d'une seconde, l'expression écrite par cette dernière est placée immédiatement après la première. La procédure d'affichage WRITELN (<VARIABLE> {VARIABLE>}) ; provoque de plus un retour du chariot et un passage à la ligne, de manière à ce que l'affichage suivant commence à la ligne suivante.

3.4.2 Mise au format des affichages

Il est possible de formater les nombres par deux paramètres de contrôle associés à la procédure WRITE.

WRITE ou WRITELN (<VARIABLE> : <P1> :
<P2> {, <VARIABLE> : <P1> : <P2>}) ;

Les paramètres supplémentaires P1 et P2 doivent être des entiers ou encore des variables de type INTEGER. P1 est le nombre total de chiffres à afficher, les manquants étant remplacés par des zéros. Le paramètre P2 n'est à employer qu'avec des nombres réels. Il donne le nombre de chiffres après la virgule. L'emploi des instructions WRITE est illustré par les programmes 3.4.2.1 WRITEDEMO et 3.4.2.2 MICKEY MOUSE.

```
PROGRAM WRITEDEMO ;

VAR   AF1:INTEGER;
      AF2:REAL;

BEGIN
  AF1:=12345;
  AF2:=12.345;
```

```
  WRITELN(AF1);WRITELN(AF2);
  WRITELN(AF1:7);
  WRITE(AF1:9);WRITELN(AF1:9);

  WRITE(AF2:8:3);WRITELN(AF2:10:2);
  WRITE(AF2:5:1);
END.
12345
 1.23450E1
   12345
     12345       12345
  12.345         12.35
 12.3
```

Programme 3.4.2.1 : Formatage de l'affichage

```
BEGIN
WRITELN;
WRITELN('     *******                  *******');
WRITELN('    *********                *********');
WRITELN('   ***********              ***********');
WRITELN('  *************            *************');
WRITELN('   *************            *************');
WRITELN('   ************************************');
WRITELN('   ************.... ....************');
WRITELN('   ************..   ...   ..************');
WRITELN('    ******* **..   ...   ..** *******');
WRITELN('              **.. *...* ..**');
WRITELN('              **...........**');
WRITELN('            *.......***.......*');
WRITELN('           *.......***.......*');
WRITELN('          *.....................*');
WRITELN('          *......***...***......*');
WRITELN('          *......*********......*');
WRITELN('           *......*******......*');
WRITELN('            *......*****......*');
WRITELN('              *......*......*');
WRITELN('                *.........*');
WRITELN('                   *...*');
WRITELN('                    ***');

END.
```

Programme 3.4.2.2 : Mickey Mouse

3.4.3 L'instruction READ

L'instruction READ a la forme suivante :

READ ou READLN (<VARIABLE> {,<VARIABLE>}) ;

<VARIABLE> ::= <ENTIER> | <REEL> | <CARACTERE>

Le système d'exploitation UCSD présente quelques faiblesses lors de l'introduction de chiffres par le clavier. Le caractère employé pour séparer les variables est un espace (SPACE). Le programme 3.4.3 READDEMO montre ces faibleses. Mais, dans de nombreux cas, l'emploi de READLN permet d'éviter les erreurs lors de l'introduction des données.

Lorsque, au cours de l'exécution d'un programme, il est nécessaire d'introduire des données comprenant plusieurs caractères, il est particulièrement indiqué d'employer une programmation axée sur le conversationnel.

Lorsque, dans un programme, est exécutée une instruction READ, le calculateur entre dans une boucle d'attente, dans laquelle il reste jusqu'à ce qu'une touche soit pressée. Mais ceci n'est indiqué, à l'extérieur, que par le clignotement du curseur. L'exemple suivant montre une possibilité d'amélioration. Il s'agit d'introduire dans un programme les variables prédéfinies A et B.

```
WRITELN ('ENTRER A ET B') ;
WRITE ('A=') ; READLN (A) ;
WRITE ('B=') ; READLN (B) ;
```

Ce qui, en BASIC, correspond à :

```
10 PRINT "ENTRER A ET B"
20 INPUT "A=" ;: A
30 INPUT "B=" ;: B
```

Une autre possibilité de l'utilisation de READLN est l'interruption de l'exécution du programme. En particulier, les données sont très fréquemment affichées extrêmement rapidement sur l'écran. Au cas où l'on désire les examiner plus calmement, il suffit d'insérer à l'endroit adéquat une instruction READLN démunie de paramètre. Le calculateur attend alors que l'on appuie sur la touche RETURN pour continuer l'affichage.

```
PROGRAM READDEMO;

CONST VIDE='          ';
```

```
VAR ENT1:INTEGER;
    ENT2:REAL;
    ENT3:CHAR;

BEGIN
  WRITELN('ENTREES');
  READ(ENT1,ENT2,ENT3);
  WRITELN(ENT1,VIDE,ENT2,VIDE,ENT3);
END.
```

3.5 Fonctions standard

L'appel d'une fonction, en PASCAL, est réalisé par :

<VARIABLE>:=<NOM DE LA FONCTION>(<ARGUMENT>) ;

Voici quelques-unes des fonctions conventionnelles :

Y := ABS(X) ;	Cette fonction calcule la valeur absolue de X
Y := SQRT(X) ;	Calcul de la racine carrée
Y := SQR(X) ;	Calcul du carré
Y := LN(X) ;	Logarithme naturel
Y := EXP(X) ;	Fonction exponentielle
Y := SIN(X) ;	
Y := COS(X) ;	Fonctions trigonométriques
Y := ARCTAN(X) ;	(sur certains compilateurs Y := ATAN(X) ;)

D'autres fonctions seront étudiées dans les chapitres suivants.

4

Les boucles et les imbrications

C'est l'emploi de boucles de programmes, d'une part, et de branchements conditionnels, d'autre part, qui permettra l'écriture de programmes courts et efficaces.

4.1 L'instruction FOR

Lorsque l'on connaît les limites d'une boucle, il est possible de la faire exécuter sous la forme d'une boucle FOR-DO. L'instruction FOR prend la forme suivante :

FOR <VARIABLE CONTROLEE> := <VALEUR INITIALE> TO <VALEUR FINALE> DO <INSTRUCTION>

ou encore :

FOR <VARIABLE CONTROLEE> := <VALEUR INITIALE> DOWNTO <VALEUR FINALE> DO <INSTRUCTION>

La variable contrôlée ainsi que ses valeurs initiale et finale doivent être des variables simples, c'est-à-dire que, à tout index i, devra correspondre un index i+1 et un index i–1. Ceci est bien le cas des entiers, des suites de caractères ou des suites d'autre types, mais pas des nombres réels.

Le pas de la boucle est égal soit à 1, soit à – 1. Il est impossible de modifier la variable contrôlée à l'intérieur de l'instruction contrôlée.

Exemples de programmes : le programme 4.1.1 calcule la somme des N premiers entiers.

Version PASCAL :

```
PROGRAM SOMME;

VAR N,I,SOMME:INTEGER;

BEGIN
  WRITE('N=');READLN(N);
  SOMME:=0;
  FOR I:=1 TO N DO SOMME:=SOMME+I;
  WRITE('LA SOMME DE 1 A ');
  WRITELN(N,' EST ',SOMME);
END.

N=100
LA SOMME DE 1 A 100 EST 5050
```

Version BASIC :

```
10 REM SOMME
20 INPUT "N=";N
30 S=0
40 FOR I=1 TO N
50 S=S+I
60 NEXT I
70 PRINT"LA SOMME DE 1 A ";N;" EST ";S
80 END
```

Programme 4.1.1 : SOMME

Dans cet exemple, la boucle FOR ne contrôle qu'une seule instruction : SOMME := SOMME+1. Lorsqu'une boucle contient plusieurs instructions, il faut indiquer leurs limites par BEGIN et END.

Programme de démonstration 4.1.2 DÉ À JOUER

Le but est l'affichage de la somme de tous les scores qu'il est possible d'obtenir par le lancer de trois dés.

Parmi toutes les boucles FOR employées ici, seule la troisième nécessite l'utilisation de BEGIN et de END pour encadrer les instructions qui suivent. En effet, les boucles précédentes ne contrôlent

Version BASIC :

```
10 REM DE
20 N=3:NE=18:VIDE$="      "
30 DIM V(NE)
50 A=0
60 FOR W1=1 TO 6
70 FOR W2=1 TO 6
80 FOR W3=1 TO 6
90 A=A+1
100 S=W1+W2+W3
110 V(S)=V(S)+1
120 NEXT W3:NEXT W2:NEXT W1
140 PRINT"NOMBRE DES JETS POSSIBLES ";A
150 PRINT"LA DISTRIBUTION DES RESULTATS EST "
160 FOR I=N TO NE
170 PRINT I,VIDE$,V(I)
180 NEXT I
190 END
```

```
NOMBRE DES JETS POSSIBLES 216
LA DISTRIBUTION DES RESULTATS EST
    3           1
    4           3
    5           6
    6           10
    7           15
    8           21
    9           25
    10          27
    11          27
    12          25
    13          21
    14          15
    15          10
    16          6
    17          3
    18          1
```

Version PASCAL :

```
PROGRAM DE;

CONST N=3;NE=18;VIDE='    ';

VAR I,W1,W2,W3,SOMME,NOMBRE:INTEGER;
    DISTRIB:ARRAY[N..NE]OF INTEGER;

BEGIN
  FOR I:=N TO NE DO DISTRIB[I]:=0;
  NOMBRE:=0;
  FOR W1:=1 TO 6 DO
   FOR W2:=1 TO 6 DO
    FOR W3:=1 TO 6 DO
    BEGIN
      NOMBRE:=NOMBRE+1;
      SOMME:=W1+W2+W3;
      DISTRIB[SOMME]:=DISTRIB[SOMME]+1;
    END;
WRITE('NOMBRE DES JETS POSSIBLES');
WRITELN('  ',NOMBRE);
WRITELN;
WRITELN('LA DISTRIBUTION DES RESULTATS EST ');
FOR I:= N TO NE DO
WRITELN(I:3,VIDE,DISTRIB[I]:6);
END
```

Programme 4.1.2 : DES

respectivement qu'une seule instruction : c'est-à-dire la boucle qui les suit. Elles ne nécessitent donc pas de BEGIN et END. Le point-virgule terminant la ligne placée avant END ; pourrait être supprimé, puisque END réalise aussi la terminaison de l'instruction. Ce point-virgule est souvent omis lorsque END ; est précédé d'une nouvelle ligne ; ceci entraîne une erreur à la compilation.

Lorsqu'il n'y a pas prédétermination du nombre des répétitions de boucle, mais que celui-ci dépend du résultat d'un calcul (comme dans les procédés itératifs), deux instructions peuvent être employées.

4.2 L'instruction REPEAT

Celle-ci prend la forme suivante :

REPEAT <INSTRUCTION>{ ; <INSTRUCTION> }
UNTIL <CONDITION> ;

Toutes les instructions placées entre REPEAT et UNTIL sont répétées jusqu'à ce que la condition soit réalisée. Le test n'étant réalisé qu'à la fin de la boucle, celle-ci est exécutée au moins une fois. La figure 4.1 l'illustre sous la forme d'un organigramme.

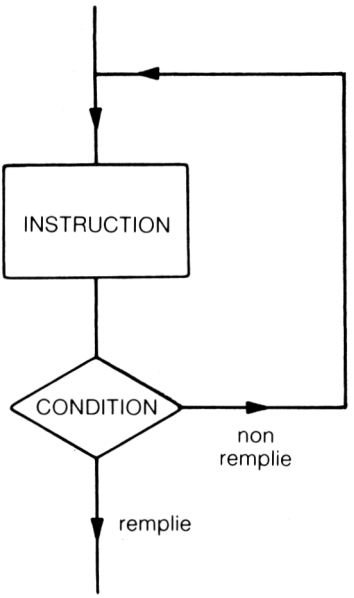


Fig. 4.1. – Diagramme REPEAT... UNTIL.

Programme de démonstration 4.2.1 FACTORIELLE

Ce programme calcule le nombre N dont la factorielle (1.2.3.4.N) est strictement supérieure à 10000.

Lorsque l'on programme une condition de fin de boucle, il ne faut pas oublier de vérifier si cette condition peut effectivement être réalisée. Par exemple, si l'on emploie une égalité, comme dans UNTIL K= 10, il faut vérifier que K puisse bien, un jour, être égal à 10. Si K est un réel, il est tout à fait possible que cette condition ne

soit jamais réalisée par suite des arrondis successifs, et l'on obtient alors une boucle infinie. Pour cette raison, il est plus sûr d'employer des inégalités.

Version PASCAL :

```
PROGRAM FACTORIELLE;
VAR J,FAK:INTEGER;
BEGIN
  FAK:=1;J:=1;
  REPEAT
    J:=J+1;
    FAK:=FAK*J
  UNTIL FAK>10000;
  WRITELN('LE NOMBRE EST ',J);
END
```

Version BASIC :

```
10  REM FACTORIELLE
20 F=1:J=0
30 J=J+1
40 F=F*J
50  IF F < 10000 THEN GOTO 30
60  PRINT "LE NOMBRE EST ";J
70  END
```

Programme 4.2.1 : FACTORIELLE

4.3 L'instruction WHILE

Une seconde possibilité de construire des boucles est donnée par l'instruction WHILE, sous la forme :

WHILE <CONDITION> DO <INSTRUCTION>

Cette fois-ci, c'est avant l'entrée en boucle que la condition est testée (fig. 4.2). Au cas où plusieurs instructions suivraient DO, il est nécessaire de les encadrer par BEGIN et END ;

Programme de démonstration 4.3.1 SUITE

On cherche à calculer la somme de la suite (1 + 1/2 + 1/3 +... + 1/N), pour N donné.

Les conditions de sortie de boucle sont, en fait, des expressions booléennes. Il est donc possible de les combiner par des opérateurs booléens ; ce sont la fonction ET (AND), la fonction OU (OR) et la négation NON (NOT).

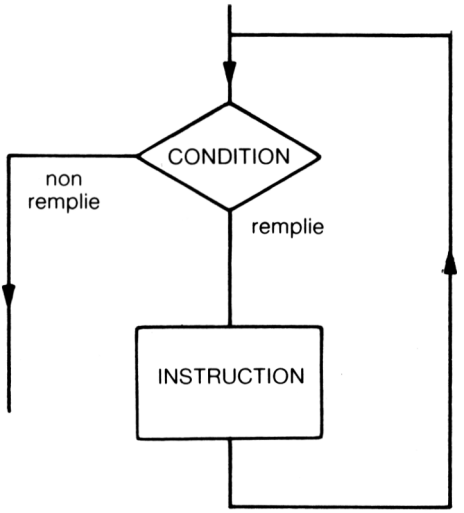


Fig. 4.2. – Diagramme WHILE.

Exemples :

WHILE ((A>10) AND (B<5)) DO...

ou

REPEAT...UNTIL ((SOMME >100) OR (CARACTERE ='E')) ;

Version PASCAL :

```
PROGRAM SUITE;
VAR N:INTEGER;
    SOMME:REAL;
BEGIN
  WRITE("N=");READLN(N);
  SOMME:=0;
  WHILE N>0 DO
    BEGIN
      SOMME:=SOMME+1/N;
      N:=N-1;
    END;
  WRITELN("SOMME=",SOMME);
  END
N=20
SOMME=3.59774
```

Version BASIC :

```
10  REM SUITE
20  INPUT "N=";N
30 S=0
40  FOR I=1 TO N
50 S=S + 1 / I
60  NEXT I
70  PRINT "SOMME=";S
80  END
```

Programme 4.3.1 : SUITE

Les fonctions booléennes sont décrites à la figure 4.3.

Le compilateur UCSD nécessite l'utilisation de parenthèses, car, pour celui-ci, les conditions de sortie de boucle doivent tenir en une seule expression. En vue d'optimiser le temps de calcul, il est préférable de simplifier au maximum les conditions de sortie de boucles, celles-ci étant recalculées à chaque passage.

A	B	A AND B
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

FONCTION ET

A	B	A OR B
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

FONCTION OU

A	NOT A
TRUE	FALSE
FALSE	TRUE

FONCTION NON

Fig. 4.3. – Fonctions booléennes.

4.4 L'instruction conditionnelle IF...THEN...ELSE

Les branchements de programme sont réalisés au moyen de l'instruction conditionnelle. Elle peut revêtir deux formes :

IF <CONDITION> THEN <INSTRUCTION>

ou

IF <CONDITION> THEN <INSTRUCTION> ELSE
<INSTRUCTION>

L'organigramme de cette instruction est représenté figure 4.4.

Si plusieurs instructions sont à exécuter après THEN et ELSE, il faut les encadrer par BEGIN et END. Lorsque, après END, suit un ELSE, il ne **faut pas** placer de point-virgule après ce END ; en effet, l'instruction, dans cette configuration, n'est pas encore terminée.

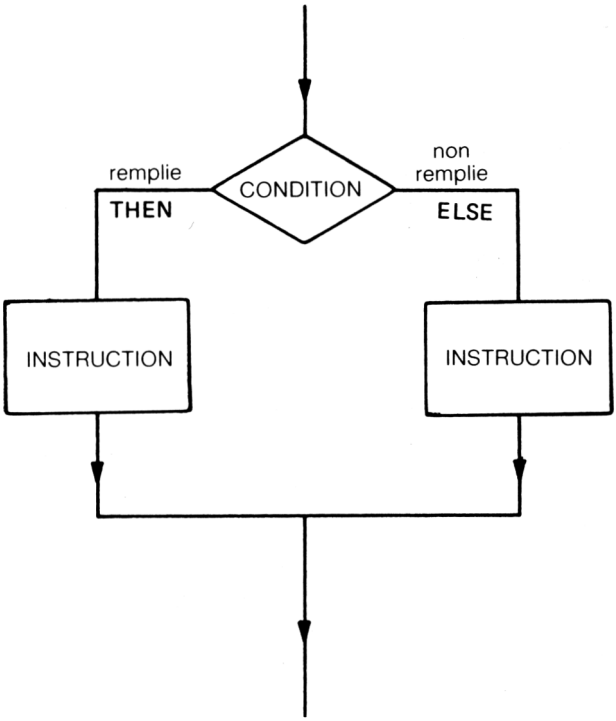


Fig. 4.4. – Diagramme IF... THEN... ELSE.

Exemple :

IF <CONDITION> THEN BEGIN <INSTRUCTION>
{ ; <INSTRUCTION> } END
ELSE BEGIN <INSTRUCTION> { ; <INSTRUCTION> } END ;

Programme de démonstration 4.4.1 CHIFFREROM

Ce programme demande un nombre entier compris entre 1 et 2000, pour le transformer en chiffres romains.

```
PROGRAM NBREROM;

CONST VIDE='     ';

VAR NBRE:INTEGER;

BEGIN
  WRITE('NBRE=');READ(NBRE);WRITE(VIDE);
  IF NBRE>3000 THEN
  WRITELN("IMPOSSIBLE")ELSE
  BEGIN
  IF NBRE>=1000 THEN BEGIN
                       WRITE('M');
                       NBRE:=NBRE-1000
                     END;
  IF NBRE>=900 THEN BEGIN
                       WRITE('CM');
                       NBRE:=NBRE-900
                     END;
  IF NBRE>=500 THEN BEGIN
                       WRITE('D');
                       NBRE:=NBRE-500
                     END;
  IF NBRE>=400 THEN BEGIN
                       WRITE('CD');
                       NBRE:=NBRE-400;
                     END;
  WHILE NBRE>=100 DO BEGIN
                       WRITE('C');
                       NBRE:=NBRE-100;
                     END;
```

```
  IF NBRE>= 90 THEN BEGIN
                       WRITE('XC');
                       NBRE:=NBRE-90
                    END;
  IF NBRE>= 50 THEN BEGIN
                       WRITE('L');
                       NBRE:=NBRE-50
                    END;
  IF NBRE>= 40 THEN BEGIN
                       WRITE('XL');
                       NBRE:=NBRE-40
                    END;
  WHILE NBRE>=10 DO BEGIN
                       WRITE('X');
                       NBRE:=NBRE-10;
                    END;
  IF NBRE>=  9 THEN BEGIN
                       WRITE('IX');
                       NBRE:=NBRE-9
                    END;
   IF NBRE>=  5 THEN BEGIN
                       WRITE('V');
                       NBRE:=NBRE-5
                    END;
   IF NBRE>=  4 THEN BEGIN
                       WRITE('IV');
                       NBRE:=NBRE-4
                    END;
   IF NBRE>=1 DO BEGIN
                    WRITE('I');
                 END;
                 WRITELN;
  END;
END.

NBRE=1981      MCMLXXXI
```

Programme 4.4.1 : NBREROM

4.5 L'instruction CASE...OF

Ce type d'instruction est programmé lorsque l'on désire faire dépendre l'exécution de différentes instructions de la valeur d'une variable. Elle se présente sous la forme suivante :

CASE<NOM>OF
    <CONSTANTE> {, <CONSTANTE> } : <INSTRUCTION>;
    < CONSTANTE> {, <CONSTANTE> } : <INSTRUCTION>
END ;

Notons que la seconde instruction ne se termine pas par un point-virgule. Certains compilateurs, en présence d'un point-virgule à cet endroit, réagissent en émettant de bizarres déclarations d'erreur.

La variable distributrice est à nouveau sous la forme :

<NOM> :: = <ENTIER> | <REEL> |
<LETTRE> | <AUTRE VARIABLE>

Les variables de la liste de distribution doivent être du même type que la variable distributrice.

Programme de démonstration 4.5.1 EXAMEN

Le but est d'exprimer les notes obtenues à un examen, sous forme de commentaire, les notes 5 et 6 devant être traduites par « insuffisant » ou « recalé » (1).

Le programme, après avoir trouvé le nombre adéquat dans la liste et exécuté l'instruction correspondante, saute à l'instruction immédiatement placée après END ;. Mais c'est aussi le cas lorsqu'aucune variable de la liste n'est sélectionnée. Lorsque ceci est source d'ambiguïtés dans le programme, il est nécessaire de prévoir les tests adéquats.

(1) N.d.t. : le système de notation allemand prévoit des notes échelonnées de 1 à 6, 1 étant la meilleure et 6 la plus mauvaise.

```
PROGRAM EXAMEN ;

VAR NOTE:INTEGER;

BEGIN
  WRITE('NOTE=');READ(NOTE);

  CASE NOTE OF
    1:WRITELN('EXCELLENT');
    2:WRITELN('BIEN');
    3:WRITELN('SATISFAISANT');
    4:WRITELN('PASSABLE');
    5,6:WRITELN("INSUFFISANT")
  END;(*FIN DE L INSTRUCTION CASE*)
END.
```

Programme 4.5.1 : EXAMEN

5

Données et ensembles de type particulier

5.1 Définition du type des matériaux

Jusqu'ici, il était encore très possible de comparer BASIC et PASCAL. Mais la possibilité de pouvoir définir soi-même des types de tableaux et d'ensembles, telle qu'elle est donnée par le PASCAL, réalise une extension considérable de la programmation.

Les exemples de déclarations de tableaux du chapitre 3.2.1 peuvent aussi être définis de la manière suivante :

Appelons notre nouveau type de données par un nom, par exemple VECTEUR, se rapportant à un tableau de un à trois nombres :

```
TYPE VECTEUR = ARRAY [1...3] OF REAL ;
```

La déclaration de type doit être placée dans la section de déclaration, avant les déclarations des variables, ce qui nous permettra d'y déclarer des variables de type VECTEUR, par exemple :

```
E1, E2, E3 : VECTEUR ;
```

Nous pouvons continuer sur notre lancée et introduire encore un nouveau type de données : le type MATRICE.

```
TYPE VECTEUR = ARRAY [1..3] OF REAL ;
     MATRICE = ARRAY [1..3] OF VECTEUR ;
VAR  E1, E2, E3:VECTEUR ;
     A, B: MATRICE ;
```

L'on peut donc effectuer les affectations suivantes :

```
E1 [1] := 1.0 ;
E1 [2] := 1.0 ;  Affectation des éléments du vecteur E1
E1 [3] := 1.0 ;
```

E2:=E1 ; E1 et E2 sont du même type, ce qui permet une affectation directe

A[1]:=E2 ; Les éléments du tableau A sont aussi du type vecteur, ce qui permet, ici aussi, l'affectation directe.

A:=B Naturellement, il est aussi possible d'affecter directement les tableaux A et B (ce sont des matrices)

Il ne faut évidemment pas en conclure que les opérations mathématiques classiques soient autorisées. Nous n'avons déclaré de règles de calcul ni pour les vecteurs ni pour les matrices. Les instructions de la forme B:=1/A, lorsque A et B sont du type matrice, restent illégales.

```
PROGRAM SECRET;

CONST VIDE='  ';

TYPE MOT=ARRAY [1...20] OF CHAR;
     CODE=ARRAY ['A'...'Z'] OF CHAR;

VAR BU,CH:CHAR;
    ENTREE,SORTIE:MOT;
    SECRET:CODE;
    I,J:INTEGER;

BEGIN
WRITELN('TEXTE:');WRITELN;

  FOR I:=1 TO 20 DO
  BEGIN
  ENTREE[I]:=VIDE;
  SORTIE[I]:=VIDE;
  END;
  CH:='Z';
  FOR BU:='A' TO 'Z' DO
  BEGIN
    SECRET[BU]:=CH;
    CH:=PRED(CH);
  END;
```

```
    I:=1;
    REPEAT
      READ(BU);
      ENTREE[I]:=BU;
      IF NOT EOLN THEN
      SORTIE[I]:=SECRET[BU];
      I:=I+1;
    UNTIL EOLN;

    FOR J:=1 TO I-1 DO WRITE(ENTREE[J]);
    WRITE(' CORRESPOND A ');
    FOR J:=1 TO I-1 DO WRITE(SORTIE[J]);

    WRITELN;
END.

TEXTE:
MONTROUGE CORRESPOND A NLMGILFTV
```

Programme 5.1.1 : SECRET

Ceci est montré par un programme non-mathématique, destiné au codage de mots. L'on a déclaré les types de variables suivants : MOT, un tableau de caractères d'une longueur de 20 lettres, et CODE, un tableau formé des lettres de l'alphabet de A à Z. Cet ensemble est connu du compilateur (voir 5.3). Les variables du type MOT sont ENTREE et SORTIE, celle du type CODE est SECRET.

Les éléments du tableau SECRET sont formés des lettres de l'alphabet, mais en ordre inverse, c'est-à-dire que SECRET [A] : = 'Z' ; SECRET [B] : = 'Y', etc. Les lettres introduites sont affectées au mot ENTREE, puis sont transférées en code sur le mot SORTIE, par appel du BU-ième élément, et ceci, jusqu'à lecture d'un caractère CR (retour du chariot). EOLN est une grandeur de type booléen qui, à la lecture de CR, prend la valeur TRUE.

D'une manière générale, la déclaration de type, pour les tableaux, prend la forme suivante :

TYPE <NOM> = ARRAY <LIMITES DES INDEX>
OF <TYPES DES ELEMENTS>

5.2 Le type RECORD

Les tableaux déclarés jusqu'à présent ne comportaient que des éléments de type homogène, c'est-à-dire soit des nombres entiers, soit des réels, soit des caractères. Le type RECORD, par contre, accepte des éléments de plusieurs types différents.

Examinons, par exemple, le fichier d'un médecin, contenant à la fois le nom, le prénom, l'âge, l'adresse, la caisse de remboursement, la date du dernier rendez-vous et d'autres renseignements encore : ce genre de fichier peut être déclaré sous RECORD.

Exemple : fichier médical.

```
TYPE CHAINE=ARRAY [1..20] OF CHAR;

     DATE=RECORD
          JOUR:1..31;
          MOIS:1..12;
          ANNEE:1900..2000
          END;

     FICHIER=RECORD
               NOM,PRENOM:CHAINE;
               ADRESSE=RECORD
                         CODEPOSTAL:INTEGER;
                         VILLE:CHAINE
                         END; (*ADRESSE*)
             AGE:0..120;
             CAISSE:'G','P';
             DR-VOUS:DATE
             END; (*FICHIER*)
```

Programme 5.2.1 : FICHIER MEDICAL

L'affectation des valeurs aux éléments de ce tableau peut s'effectuer sous la forme suivante :

FICHIER.NOM: = 'DUPONT' ;
FICHIER.PRENOM: = 'ROBERT' ;
FICHIER.CODE POSTAL: = 75300 ;
FICHIER.VILLE: = 'PARIS' ;

Un tel travail d'écriture peut être considérablement simplifié par l'emploi de l'instruction WITH.

5.2.1 L'instruction WITH

Sa forme est :

WITH <NOM DU FICHIER RECORD> {, <NOM DU FICHIER RECORD>} DO <INSTRUCTION>

Ce qui permet d'écrire l'affectation de l'exemple suivant sous une forme simplifiée :

```
WITH FICHIER,ADRESSE,DATE DO
BEGIN
    NOM: = 'DUPONT' ;
    PRENOM: = 'ROBERT' ;
    CODE POSTAL: = 75300 ;
    VILLE: = 'PARIS' ;
    AGE: = 56 ;
    CAISSE: = 'G' ;
    JOUR: = 15 ;
    MOIS: = 5 ;
    ANNEE: = 1980
END ;
```

Des exemples de programmes suivront dans les chapitres suivants. La forme de la déclaration d'un tableau RECORD est la suivante :

TYPE <NOM DU TABLEAU> = RECORD
<LISTE DES ELEMENTS> END ;

<LISTE DES ELEMENTS> :: = <NOM>
{, <NOM>} : <TYPE> :
<NOM> {, <NOM>} : <TYPE> ;

5.3 Les ensembles et leurs dénombrements

Les ensembles simples se définissent par l'énumération de leurs éléments. Nous connaissons déjà un ensemble : celui des valeurs booléennes TRUE et FALSE, qui, d'ailleurs, peut aussi être déclaré par :

TYPE BOOLEAN = (FALSE, TRUE) ;

L'on peut, de la même façon, définir soi-même de nouveaux ensembles :

TYPE JOURS = (LU, MA, ME, JE, VE, SA, DI) ;
ou
TYPE AUTO = (OPEL, FIAT, MERCEDES, PEUGEOT, RENAULT) ;

A ces ensembles s'appliquent les fonctions suivantes :

A) ORD détermine l'ordinal.

Exemple : ORD(ME)=2,ORD(LU)=0.

Le premier élément d'un ensemble a donc toujours l'ordinal 0. Comme les éléments forment une suite ordonnée, il est possible de leur appliquer les opérateurs de comparaison et de les employer en tant que paramètres dans les boucles FOR : un exemple d'application a été vu au programme 5.1.1 sur l'ensemble des lettres de l'alphabet.

B) PRED détermine l'élément précédent de la suite. Exemple : PRED (MERCEDES) = FIAT, dans l'ensemble des marques de voitures déclaré plus haut.

C) SUCC détermine l'élément successeur. SUCC(VE)=SA.

L'affectation d'une variable à un ensemble est effectué par l'instruction :

SET OF

Exemple :

```
TYPE ANNEE = (JAN, FEV, MAR, AVR, MAI, JUN, JUL, AOU, SEP,
OCT, NOV, DEC) ;
VAR VACANCES : SET OF ANNEE ;
    HIVER : SET OF ANNEE ;
```

Nous pouvons maintenant affecter des éléments de l'ensemble ANNEE aux variables VACANCES et HIVER :

```
VACANCES:= [JUL, AOU] ;
HIVER:= [JAN..MAR, OCT..DEC] ;
```

En PASCAL, l'intégralité des caractères, c'est-à-dire les lettres majuscules et minuscules, les nombres et les caractères spéciaux sont déclarés sous SET OF CHAR :

```
VAR ALPHABET, VOYELLES, CONSONNES : SET OF CHAR ;
ALPHABET:= ['A'..'Z'] ;
VOYELLES:= ['A','E','I','0','U'] ;
CONSONNES:= ALPHABET – VOYELLES ;
```

Le signe « moins » de la dernière instruction est un opérateur employé avec les ensembles ALPHABET et VOYELLES. Il donne la différence entre deux ensembles. L'union de deux ensembles est donnée par le signe « plus » :

```
ALPHABET:= CONSONNES + VOYELLES ;
```

L'intersection de deux ensembles est donnée par le signe « multiplié » :

['A', 'E', 'I','O'] * ['A','B','C','D','E']=['A','E']

L'opérateur spécifique IN a la particularité de constater si une certaine variable est, ou non, un élément d'un ensemble donné.

La question :

IF CARACTERE IN ['+','–','*','/'] THEN...

donne TRUE, si la variable est bien contenue dans cet ensemble, et FALSE dans les autres cas.

Programme de démonstration 5.3.1 RESULTAT

L'on introduit les notes obtenues à un examen : celui-ci est considéré comme étant réussi pour les notes de 1 à 4 (les notes variant de 1 à 6). Le résultat de l'examen est affiché sous la forme du pourcentage des réussites par rapport au nombre total de participants. L'introduction des notes se prolonge, jusqu'à ce que l'on tape NOTE = ZERO. L'introduction de caractères non contenus dans l'ensemble NOTES déclenche l'affichage de « DONNEE INCORRECTE ».

```
PROGRAM RESULTAT;

VAR NOTES,REUSSITE:SET OF CHAR;
    I,PARTICIPANTS,ECHEC:INTEGER;
    MOYENNE,POURCENT:REAL;
    NOTE:CHAR;

BEGIN
  WRITELN('DONNEES');
  NOTES:=('1'..'6');
  REUSSITE:=('1'..'4');
  PARTICIPANTS:=0;ECHEC:=0;I:=0;

  REPEAT
    READ(NOTE);WRITELN;
    IF NOTE IN NOTES THEN
    BEGIN
      I:=I+ORD(NOTE)-48;
      PARTICIPANTS:=PARTICIPANTS+1;

      IF NOT (NOTE IN REUSSITE) THEN
        ECHEC:=ECHEC+1;
    END
```

```
    ELSE IF NOTE <> '0' THEN
    WRITELN('DONNEES INCORRECTES!');
  UNTIL NOTE='0';
  POURCENT:=(PARTICIPANTS-ECHEC)/PARTICIPANTS*100;
  MOYENNE:=I/PARTICIPANTS;
  WRITELN;
  WRITELN('RESULTAT DE L EXAMEN:');
  WRITELN;
  WRITELN('NOMBRE DE PARTICIPANTS: ',PARTICIPANTS);
  WRITELN('MOYENNE : ',MOYENNE:6:2);
  WRITELN('REUSSITES : ',POURCENT:6:2,'%');
END
```

Programme 5.3.1 : RESULTAT

6

Procédures et fonctions

6.1 Procédures

Une procédure est une partie de programme autonome, construite comme tout autre programme. Elle possède donc un en-tête, une section de déclaration et un programme principal.

L'en-tête est formé du nom de la procédure et d'une liste des paramètres, celle-ci pouvant être vide.

PROCEDURE <NOM> (<LISTE DES PARAMETRES>) ;

Occupons-nous tout d'abord de la liste des paramètres.

Prenons l'exemple suivant :

Soit à écrire une procédure pour l'affichage d'un texte de longueur indifférente, mais inférieure à 80 caractères. Il faut donc qu'à l'appel de cette procédure, soient transmis à la fois le texte en lui-même, et sa longueur. Ces valeurs, que l'on inscrira dans la liste des paramètres, sont appelés des paramètres **formels.**

```
PROCEDURE TEXTE AFF (TXT : ARRAY [1..80] OF CHAR ;
L :INTEGER) ;
VAR J :INTEGER ;

BEGIN
FOR J : = 1 TO L DO WRITE TXT[J] ;
WRITELN ;
END ;(*TEXTEAFF*)
```

Nous allons utiliser cette procédure dans le petit programme 6.1.1 TEXTDEM01.

Ce programme lit deux phrases (PHR1, PHR2), puis les affiche à nouveau par la procédure TXTSORT.

Au premier appel de la procédure, les paramètres transmis par la liste des paramètres sont PHR1 et I. Ce sont les paramètres

```
PROGRAM TEXTDEMO1;

CONST VIDE=' ';

TYPE CARTE = ARRAY[1..80] OF CHAR;

VAR PHR1,PHR2:CARTE;
    I,J:INTEGER;
    CH:CHAR;

PROCEDURE TEXTSORT(TXT:CARTE;L:INTEGER);

VAR I:INTEGER;
BEGIN
  FOR I:=1 TO L DO WRITE(TXT[I]);
  WRITELN;
END; (*TEXTSORT*)

BEGIN (*PR PRINCIPAL*)
  WRITELN('DONNEES: ');WRITELN;
  I:=0;
  REPEAT
    READ (CH);
    IF NOT EOLN THEN
    BEGIN
      I:=I+1;
      PHR1[I]:=CH
    END;
  UNTIL EOLN OR (I=80);

  J:=0;
  REPEAT
    READ (CH);
    IF NOT EOLN THEN
    BEGIN
      J:=J+1;
      PHR2[J]:=CH;
    END;
  UNTIL EOLN OR (J=80);
```

```
  (*AFFICHAGE DU TEXTE*)
WRITELN('CECI EST LA 1.PHRASE: ');
TEXTSORT(PHR1,I);
WRITELN('CECI EST LA 2.PHASE: ');
TEXTSORT(PHR2,J);

END.
```

Programme 6.1.1 : TEXTDEMO1

actuels, valides lors de l'appel de la procédure. Au second appel, les paramètres actuels sont PHR2 et J.

Il y a donc une distinction à faire entre les **paramètres formels,** lesquels sont définis au sein de la liste des paramètres lors de la déclaration de la procédure, et les **paramètres actuels,** qui sont transmis par la liste des paramètres à l'appel de la procédure. Naturellement, il faut que les paramètres formels et actuels soient du même type.

Nous désirons maintenant transformer l'étape d'introduction du texte en une autre procédure, que nous appellerons TXTENT. Nous désirons conserver les mêmes paramètres TXT et L. Ces paramètres formels devront être modifiés par la procédure, car la longueur L du texte n'est pas connue à l'avance. Ils seront donc déclarés en tant que variables (VAR), dans la liste des paramètres.

PROCEDURE TXTENT (VAR TXT : CARTE : VAR L : INTEGER) ;

Ce qui permet de modifier le programme 6.1.1 de la manière suivante (programme 6.1.2 TEXTDEM02) :

L'on aura remarqué que, dans les deux programmes ci-dessus, la variable I est déclarée aussi bien dans le programme principal que dans les procédures. Quelle peut être la valeur de cette variable dans les différentes parties du programme ? La réponse à cette question débouche sur la notion de programmation structurée et sur la distinction entre variables **globales** et **locales.** Comme nous l'avons déjà mentionné, les procédures sont des blocs de programme indépendants, ne se différenciant d'un programme véritable que par la liste des paramètres. Ainsi, les variables déclarées au sein des procédures ne seront reconnues qu'à l'intérieur de celles-ci. L'on nomme donc ces variables des variables **locales.**

Les variables déclarées dans la section de déclaration, par contre, sont des variables **globales,** et sont reconnues tant dans le programme principal que dans les procédures. Dans notre cas, le

programme principal et une procédure comportent une déclaration de variable de même nom, et, pourtant, ce sont deux grandeurs différentes. Si cette variable prend une certaine valeur au sein d'une

```
PROGRAM TEXTDEMO2;

CONST VIDE=' ';

TYPE CARTE = ARRAY [1..80] OF CHAR;

VAR PHR1,PHR2:CARTE;
    I,J:INTEGER;
    CH:CHAR;

PROCEDURE TEXTSORT(TXT:CARTE;L:INTEGER);

VAR I:INTEGER;
BEGIN
  FOR I:=1 TO L DO WRITE(TXT[I]);
  WRITELN;
END; (*TEXTSORT*)

PROCEDURE TEXTENT(VAR TXT:CARTE;VAR L:INTEGER);

VAR I:INTEGER;

BEGIN
  I:=0;
  REPEAT
    READ (CH);
    IF NOT EOLN THEN
    BEGIN
      I:=I+1;
      TXT[I]:=CH;
      L:=I;
    END;
  UNTIL EOLN OR (I=80);
END;  (*TEXTENT*)

BEGIN (*PR PRINCIPAL*)
  WRITELN ('ENTRER LA 1.PHRASE:');
  TEXTENT (PHR1,I);
  WRITELN;
  WRITELN ('ENTRER LA 2.PHRASE:');
  TEXTENT (PHR2,J);
  WRITELN;
  WRITELN ('AFFICHAGE DES DEUX PHRASES:');
  TEXTSORT(PHR1,I);TEXTSORT(PHR2,J);
END.
```

Programme 6.1.2 : TEXTDEMO2

procédure, cela ne **modifiera pas** pour autant la valeur d'une autre variable de même nom, appartenant à une autre procédure ou même au programme principal. Cette propriété, difficile à admettre, surtout pour les programmeurs habitués au BASIC, est illustrée par le programme suivant :

```
PROGRAM VARDEMO;

CONST VIDE=' ';

TYPE NOM=ARRAY[1..10]OF CHAR;

VAR A,B:NOM;

PROCEDURE VIDER(VAR C:NOM);
VAR I:INTEGER;
BEGIN
  FOR I:=1 TO 10 DO C[I]:=VIDE;
END; (*VIDER*)

PROCEDURE ENT(VAR C:NOM);
VAR I:INTEGER;
    CH:CHAR;
BEGIN
  I:=0;
  VIDER(C);
  REPEAT
    READ(CH);
    IF NOT EOLN THEN
    BEGIN
      I:=I+1;
      C[I]:=CH;
    END;
  UNTIL EOLN OR (I=10);
END; (*ENT*)

PROCEDURE SORT(C:NOM);
VAR I:INTEGER;
BEGIN
```

```
  FOR I:=1 TO 10 DO WRITE(C[I]);
END; (*SORT*)

PROCEDURE DEMO (C:NOM;VAR D:NOM);
VAR A:NOM;
BEGIN
  A:=C;SORT(A);
  C[1]:='*';A:=C;SORT(A);WRITELN;
  A:=D;SORT(A);
  D[1]:='*';A:=D;SORT(A);WRITELN;
END; (*DEMO*)

BEGIN  (*PR PRINCIPAL*)
  WRITE('ENTRER A:');
  ENT(A);WRITELN;
  WRITE('ENTRER B:');
  ENT(B);WRITELN;

  DEMO(B,A);WRITELN;

  WRITELN('A ET B SONT MAINTENANT:');
  WRITELN;
  SORT(A);SORT(B);
END.


ENTRER A:DUPONT
ENTRER B:DURAND
DUPONT       *UPONT
DURAND       *URAND

A ET  B SONT MAINTENANT:

*UPONT     DURAND
```

Programme 6.1.3 : VARDEMO

Ce programme affecte deux noms aux variables A et B. Tous deux sont transmis à la procédure DEMO, l'un pouvant être modifié, l'autre restant invariable. Cette procédure comporte à nouveau une

déclaration de variable A, à laquelle sont affectés les deux noms ; puis ces noms sont affichés, de manière non modifiée, d'une part, et de manière modifiée, d'autre part.

Si l'on faisait afficher les deux noms après avoir quitté le sous-programme, on constaterait que le nom transmis en tant que variable est modifié, tandis que l'autre reste tel quel. D'autre part, il ressort que la variable A, employée deux fois, possède, tout en gardant le même nom, deux valeurs différentes, suivant que l'on se trouve à l'intérieur ou à l'extérieur de la procédure.

Ce mode de gestion des variables permet aussi d'employer ce que l'on appelle les procédures **récursives.** Ce sont des sous-programmes capables de s'appeler eux-même. Nous en reparlerons quand nous verrons les fonctions récursives.

En attendant, quelques conseils sur la technique de programmation et sur la structure des programmes en PASCAL.

Les procédures sont d'une grande utilité en programmation. Les problèmes complexes peuvent ainsi être subdivisés en plusieurs problèmes plus simples, que l'on ne définira tout d'abord que par leur nom et leur rôle.

Prenons un exemple : soit un programme de gestion du fichier des adhérents d'un club. Ce programme devra se charger, entre autres, des tâches partielles suivantes :

l'enregistrement de nouveaux adhérents
le départ d'adhérents
les distinctions
la trésorerie.

Globalement, l'on déclarera un fichier ADHERENTS et, par exemple, un sous-programme ENREGISTREMENT pour lequel les paramètres transmis seront rassemblés en un fichier RECORD contenant les données d'un adhérent. Il est possible de développer et de tester cette partie de programme tout à fait indépendamment du programme principal. Mieux : il est possible de faire travailler plusieurs programmeurs sur le même programme, chacun d'eux étant totalement libre de choisir les noms de variables qu'il désire. Il leur suffira de respecter les types des paramètres.

Cette façon de concevoir les programmes est aussi appelée :

TOP-DOWN APPROACH

Le problème est ici traité de haut en bas, de plus en plus profondément. Les informaticiens mal intentionnés parlent ici de « repousser continuellement les difficultés ».

Le fait de subdiviser les programmes en programmes partiels donne un autre avantage : ces programmes partiels peuvent être consignés au sein d'une bibliothèque de logiciel, pour être, en cas de besoin, intégrés à de nouveaux programmes. Comme les dénominations des variables ne se recoupent pas, les modifications des programmes et la conception de programmes nouveaux en sont grandement facilitées.

De cette manière, les programmes, même compliqués, se construisent de façon claire et économique en temps et en argent. C'est un point important en ce qui concerne la conception du logiciel commercial.

6.2 Les fonctions

L'on déclare les fonctions de la manière suivante :

FONCTION <NOM> (<LISTE DES PARAMETRES>) : <TYPE> ;

<NOM> est le nom de la fonction ; il servira à appeler celle-ci au cours du programme.

<LISTE DES PARAMETRES> est, comme pour les procédures, la liste des paramètres formels transmis à la fonction.

<TYPE> indique de quel type est la valeur retransmise au programme par la fonction.

Lors de l'appel d'une fonction, par son nom, celle-ci affecte une valeur à une variable devant être du même type que cette fonction.

Exemples :

```
FONCTION CUBE (X : REAL) : REAL ;
BEGIN
CUBE : =x.x.x ;
END ;
```

Cette fonction élève au cube la valeur de la variable X.

```
FONCTION MIN (a,b : INTEGER) : INTEGER ;
BEGIN
IF a<b  THEN MIN : =a
        ELSE MIN : =b ;
END ;
```

La fonction MIN détermine la plus petite des deux valeurs a et b.

```
FONCTION OK (MIN, UTIL, MAX : REAL) : BOOLEAN ;
BEGIN
```

```
IF (UTIL≤MAX) OR (UTIL≥MIN)  THEN OK := TRUE
                             ELSE OK := FALSE ;
END ;
```

La fonction OK transmet TRUE lorsque la variable UTIL (résultant d'une mesure, par exemple) est comprise entre les limites MIN et MAX.

6.3 Procédure et fonctions récursives

L'emploi de variables locales, n'étant utilisées que lors de l'appel d'une procédure ou d'une fonction, tout en étant connues du programme, permet la réalisation d'algorithmes **récursifs.** On appelle récursivité le processus de calcul d'un résultat à partir de résultats partiels provenant de calculs identiques précédents.

Exemple :

5 ! = 1.2.3.4.5. Soit à calculer 6 !. On peut le faire soit en calculant 6 ! = 1.2.3.4.5.6, soit en posant 6 ! = 5 !.6 ; la formule générale étant (N+1) ! = N !.(N+1). Cela nous permet de calculer la valeur de la fonction pour N+1 à partir de sa valeur pour N. Le programme 6.3.1 RECURFAC en est un exemple. Dans de nom-

```
PROGRAM RECURFAC;

VAR N:INTEGER;

FUNCTION FAC(N:INTEGER):INTEGER;

BEGIN
    IF N=0 THEN FAC:=1 ELSE FAC:=FAC(N-1)*N;
END;  (*FAC*)

BEGIN (*PR PRINCIPAL*)
WRITE('N=');READLN(N);
WRITELN('N!=',FAC(N));
END.
```

Programme 6.3.1 : RECURFAC

breux cas, il est possible d'éviter l'emploi de procédés récursifs en programmant de façon linéaire. Il y a cependant des cas où la programmation récursive est indispensable, notamment en stratégie.

Ces procédés ont un autre inconvénient : ils consomment beaucoup de place mémoire. A chaque nouvel appel, toutes les variables employées réservent un espace supplémentaire. Ceci peut même mener à un dépassement de capacité, si les variables déclarées dans une procédure récursive se trouvent être des tableaux.

L'exemple pris pour les procédures récursives sera le procédé de tri rapide inventé par HOARE (QUICKSORT), qui est très représentatif.

```
PROGRAM QUICKSORT;
CONST N=1000;
VAR I,Z:INTEGER;
    A:ARRAY[1..N] OF INTEGER;

PROCEDURE SORT(GAUCHE,DROITE:INTEGER);
VAR I,J,X,W:INTEGER;
BEGIN
  I:=GAUCHE;J:=DROITE;X:=A[(I+J)DIV 2];
  REPEAT
    WHILE A[I] < X DO I:=I+1;
    WHILE A[J] > X DO J:=J-1;
    IF I <= J THEN
    BEGIN
      W:=A[I];A[I]:=A[J];A[J]:=W;
      I:=I+1;J:=J-1;
    END;
  UNTIL I > J;
  IF GAUCHE < J THEN SORT (GAUCHE,J);
  IF I < DROITE THEN SORT (I,DROITE);
  END; (*SORT*)

BEGIN (*PR PRINCIPAL*)

FOR I:=1 TO N DO A[N+1-I]:=I;
```

```
WRITE('DEBUT DU TRI');

SORT(1,N);

WRITE('...TERMINE');WRITELN;
END.
```

Programme 6.3.2 : QUICKSORT

7

Structures de données dynamiques

Les structures de données employées jusqu'à présent sont d'un usage limité et, de plus, elles doivent être définies par leurs limites avant d'être appelées (par exemple ARRAY [1..10] OF INTEGER ;). Les structures de données dynamiques, par contre, peuvent être étendues à loisir après avoir été déclarées, comme par exemple les LISTES. Celles-ci comportent deux types de renseignements : d'une part l'enregistrement de la liste elle-même et, d'autre part, au moins un indicateur (ou pointeur) renvoyant sur l'enregistrement suivant. La valeur de cette variable indicatrice est donc l'adresse à laquelle se trouve l'enregistrement suivant. Ceci est illustré par la figure 7.1. Cette liste comprend trois enregistrements.

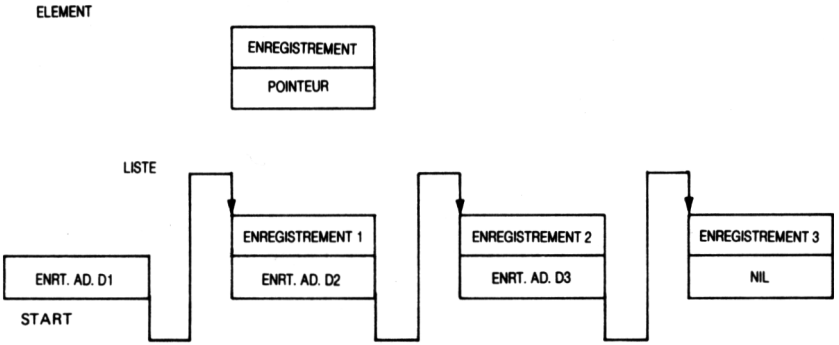


Fig. 7.1. – LISTE.

Chaque élément de la liste comprend un enregistrement et l'adresse de celui qui le suit. Le pointeur du dernier enregistrement prend la valeur NIL (NOT IN LIST), ce qui signifie : « il n'y a pas de

successeur à cet élément ». Les listes nécessitent encore la donnée suivante : le départ du pointeur, c'est-à-dire l'adresse du premier élément de liste.

7.1 Construction d'une liste

Il faut d'abord déclarer un pointeur :

```
TYPE POINTEUR = ↑ LISTE ;
```

puis la liste elle-même, en un tableau RECORD à deux éléments :

```
LISTE =  RECORD
         SUIVANT:POINTEUR ;
         ENREGIST:ARRAY [1..20] OF CHAR
         END ;
```

Le premier élément est ici du type pointeur, et contient l'adresse de l'enregistrement suivant. L'enregistrement lui-même est le second élément. Le programme aura besoin de deux autres pointeurs (également des variables) :

```
VAR PTR, DEBUT:POINTEUR ;
```

Nous pouvons maintenant effectuer le premier enregistrement sur la liste. Affichons DEBUT:=NIL;, aucun enregistrement n'étant présent pour l'instant. La fonction NEW (PTR) permettra de créer de l'espace mémoire pour les données de la LISTE, et d'affecter des valeurs aux éléments :

```
NEW (PTR)
PTR↑.ENREGIST:='HENRI DUPONT' ;
PTR↑.SUIVANT:=DEBUT ;
DEBUT:=PTR ;
```

Ces lignes sont à interpréter de la façon suivante : l'instruction NEW (PTR) attribue l'adresse d'un espace mémoire réservé pour les données sous LISTE, à la variable PTR. C'est sur ces données que pointe le pointeur PTR. Un élément est alors choisi par PTR ↑ . ENREGIST. PTR↑. renvoie donc sur les données LISTE.

SUIVANT est remplacé par DEBUT. Celui-ci vaut NIL avant le premier enregistrement, puis il lui est affecté la valeur de PTR. Le premier enregistrement étant fait, cela donne l'aspect suivant :

PTR↑.ENREGIST est HENRI DUPONT

PTR↑. SUIVANT est NIL

D'autre part, DEBUT contient maintenant l'adresse des données nouvelles enregistrées.

Examinons les pointeurs après un second enregistrement. SUIVANT contient alors l'adresse du premier enregistrement, et DEBUT contient l'adresse du second.

Notre programme construit donc une liste dont les nouveaux éléments se placent **avant** leurs prédécesseurs. L'on obtient l'affichage des enregistrements par :

```
PTR:=DEBUT ;
WHILE PTR <>NIL DO
BEGIN
WRITE (PTR↑.ENREGIST) ;
PTR:=PTR↑.SUIVANT ;
END ;
```

Ce programme transfère à PTR l'adresse du dernier enregistrement effectué, affiche la valeur de PTR↑.ENREGIST, puis donne à PTR l'adresse de l'enregistrement suivant. Ceci est répété jusqu'à ce que l'adresse devienne NIL, indiquant que le dernier (effectué en premier) enregistrement est atteint.

Le programme 7.1.1 LISTEDEMO utilise la méthode ci-dessus pour construire une liste de noms, laquelle est alors affichée.

```
PROGRAM LISTEDEMO;

CONST VIDE=' ';

TYPE PTR=^LISTE;
     NOM=ARRAY[1..20]OF CHAR;
     LISTE=RECORD
             SUITE:PTR;
             ENRT:NOM;
             END;

VAR DEBUT,Z:PTR;
    CH:CHAR;
    I:INTEGER;
```

```
BEGIN
DEBUT:=NIL;
REPEAT
  REPEAT
    WRITE('NOUVEL ENREGISTREMENT (O/N) ?');
    READ(CH);WRITELN;
    IF CH<>'N' THEN
    BEGIN
      NEW(Z);
      FOR I:=1 TO 20 DO
      IF NOT EOLN THEN
      READ(Z^.ENRT[I])ELSE
      Z^.ENRT(I):=VIDE;
      Z^.SUITE:=DEBUT;
      DEBUT:=Z;
    END;
  UNTIL CH='N';

  WRITELN;WRITE('IMPRIMER LA LISTE (O/N) ?');
  READ(CH);WRITELN;
  IF CH<>'N' THEN
  BEGIN
    Z:=DEBUT;
    WHILE Z<>NIL DO
    BEGIN
      FOR I:=1 TO 20 WRITE(Z^.ENRT(I));
      WRITELN;
      Z:=Z^.SUITE;
    END;
  END;

WRITE('CONTINUER LA LISTE (O/N) ?');
READ(CH);WRITELN;

UNTIL CH='N';

WRITELN ('FIN');
END
```

Programme 7.1.1 : LISTDEMO

7.2 Insertion et suppression au sein de listes classées

Avec le programme LISTEDEMO, l'ordre des enregistrements était prédéterminé par l'ordre de l'entrée des données. Considérons maintenant une liste susceptible d'être classée suivant un critère quelconque. L'exemple sera une liste de prix comprenant à la fois le numéro de commande, la description de l'article et son prix. Nous devons donc déclarer les données suivantes :

```
LISTEPRIX= RECORD
           NUMCOM:INTEGER ;
           ARTICLE:ARRAY [1..20] OF CHAR ;
           PRIX:REAL ;
           SUITE:POINTEUR
           END ;
```

où nous avons :

```
TYPE POINTEUR=↑LISTEPRIX
```

Soit à classer cette liste suivant les numéros de commande. La figure 7.2.1 montre l'insertion d'un nouvel enregistrement à l'intérieur d'une liste déjà classée.

Le but est d'insérer l'élément numéro 102. La liste est coupée entre 100 et 105 ; le pointeur de l'élément à insérer est remplacé par celui de son prédécesseur, lequel a son pointeur dirigé sur le nouvel élément.

La suppression d'un enregistrement s'opère simplement par échange entre les pointeurs de l'élément précédent, et de l'élément à supprimer (fig. 7.3 et programme 7.2.1 LISTEPRIX).

7.3 Fichiers séquentiels

Il existe une autre possibilité de créer des structures de données dynamiques : ce sont les fichiers sur mémoire de masse externe. En PASCAL, ces fichiers sont appelés FILE. Ils sont constitués de données du même type, c'est-à-dire ARRAY ou RECORD, disposées séquentiellement.

En général, un fichier de ce type est déclaré par :

```
TYPE <NOM> =FILE OF <TYPE>.
```

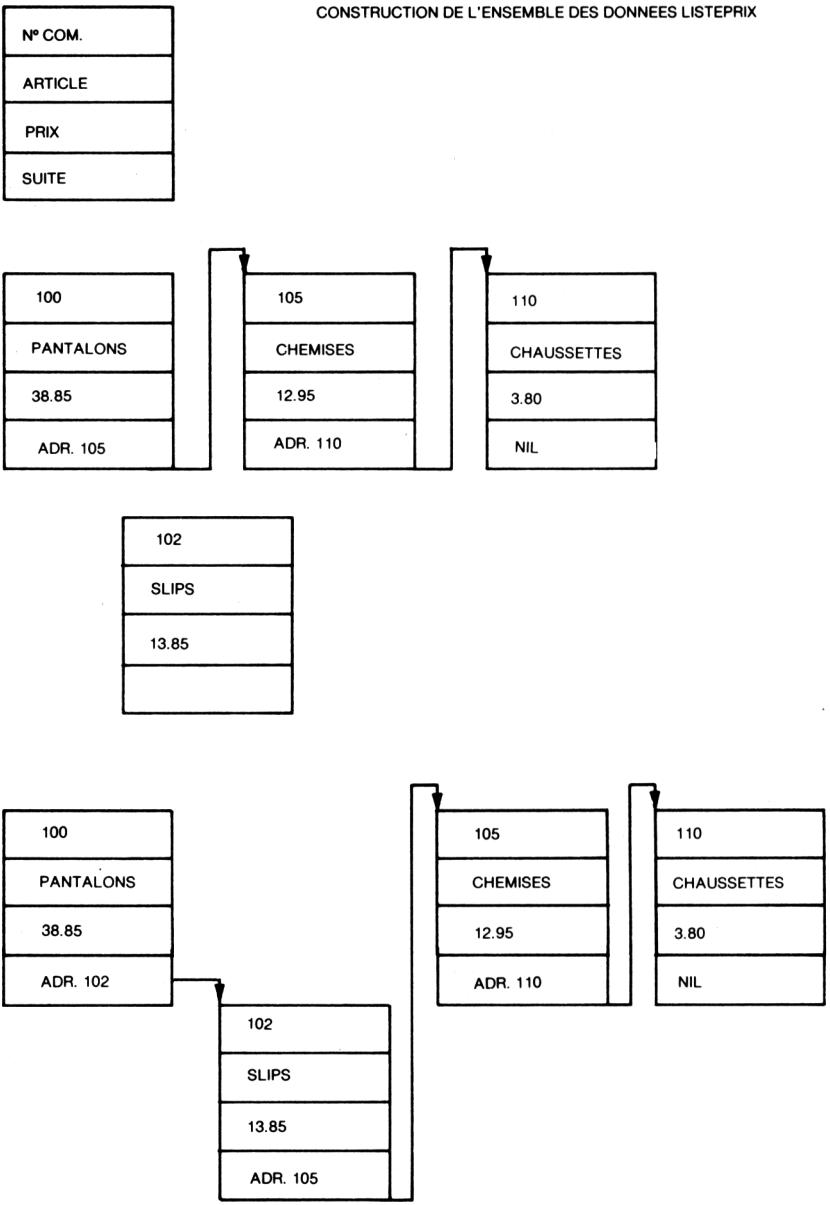


Fig. 7.2. – Insertion d'un nouvel enregistrement.

```
PROGRAM LISTEPRIX;

CONST VIDE=' ';K=20;

TYPE PTR=^PLISTE;
     DESCRIPTION=ARRAY[1..K]OF CHAR;
     PLISTE=RECORD
              NUMCOM:INTEGER;
              ARTICLE:DESCRIPTION;
              PRIX:REAL;
              SUITE:PTR
              END;
VAR DEBUT,PTR1,PTR2:PTR;
    NOMBRE:INTEGER;
    CH:CHAR;

PROCEDURE INSERTION(VAR PTR:PTR);
VAR P1,P2:PTR;
    TROUVE:BOOLEAN;

BEGIN
  TROUVE:=FALSE;
  IF DEBUT=NIL THEN
  BEGIN
    PTR^.SUITE:=DEBUT;
    DEBUT:=PTR;
    END; (*1.ELEMENT DE LA LISTE*)
  ELSE
  IF PTR^.NUMCOM<DEBUT^.NUMCOM
  THEN BEGIN
         PTR^.SUITE:=DEBUT;
         DEBUT:=PTR
       END
  ELSE BEGIN
         P2:=DEBUT;
         REPEAT
         P1:=P2^.SUITE;
         IF P1<>NIL THEN
         IF P1^.NUMCOM < PTR].NUMCOM
```

```
          THEN P2:=P1
          ELSE BEGIN
                 P2^.SUITE:=PTR;
                 PTR^.SUITE:=P1;
                 TROUVE:=TRUE
               END;
          UNTIL(P1=NIL)OR TROUVE;
          IF P1=NIL THEN BEGIN
                           P2^.SUITE:=PTR;
                           PTR^.SUITE:=NIL;
                         END;
          END;
END;  (*INSERTION*)

PROCEDURE ENTREE(VAR P:PTR);
VAR N,I:INTEGER;
    P:REAL;
BEGIN
  NEW(Z);
  WRITE('NUMERO DE COMMANDE: ');READLN(N);
  Z^.NUMCOM:=N;
  WRITE('ARTICLE: ');
  FOR I:=1 TO K DO
   IF NOT EOLN THEN READ(Z^.ARTICLE[I])
               ELSE Z^.ARTICLE[I]:=VIDE;
  WRITELN;
  WRITE('PRIX: ');READLN(P);
  Z^.PRIX:=P;
END;  (*ENTREE*)

PROCEDURE AFFICHAGE(P:PTR);
VAR I:INTEGER;
BEGIN
  WITH Z^ DO
  BEGIN
    WRITELN('NUMERO DE COMMANDE: ',NUMCOM);
    WRITE  ('ARTICLE: ');
    FOR I:=1 TO K DO WRITE (ARTICLE[I]);
    WRITELN;
```

```
    WRITELN('PRIX      :',PRIX:8:2);
  END;
END;  (*AFFICHAGE*)

BEGIN
  DEBUT:=NIL;
  REPEAT
  WRITE('AJOUTER UN ENREGISTREMENT (O/N) ');
  READ(CH);WRITELN;
  IF CH='O' THEN
  BEGIN
    ENTREE(PTR1);
    INSERTION(PTR1);
  END;
  WRITELN;WRITE('AFFICHER LA LISTE (O/N) ');
  READ(CH);WRITELN;
  IF CH<>'N' THEN
  BEGIN
    PTR1:=DEBUT;
    WHILE PTR1<> NIL DO
    BEGIN
      AFFICHAGE(PTR1);
      READLN;
      PTR1:=PTR1^.SUITE;
    END;
  END;
  WRITE('TERMINER (O/N) ');
  READ(CH);WRITELN;
  UNTIL CH='O';
END
```

Programme 7.2.1 : LISTEPRIX

Cette déclaration crée en même temps une variable <NOM>↑, permettant l'accès aux données et leur modification.

Le transfert des données entre le calculateur et la mémoire de masse est assisté des fonctions standard suivantes :

Examinons le programme de démonstration 7.3.1 TELEPHONE.

Nous y déclarons un fichier FICHIER composé des éléments

NOM, PRENOM, NUMERO. Ce fichier sera appelé ENREGIST. Les enregistrements seront donc rassemblés en FICHIER=FILE OF ENREGIST.

La procédure

REWRITE(<NOM>,'<NOM DU FICHIER>:<ELEMENT>') ;

ouvre un nouveau fichier sur la disquette <NOM DU FICHIER>, avec le fichier <ELEMENT>.

Dans notre exemple, l'expression :

REWRITE(FICHIER,'TELE:PHONE') ;

aménage un fichier nommé PHONE, sur la disquette TELE, ce fichier recevant les données à enregistrer. L'introduction des données étant effectuée, celles-ci sont stockées sur la disquette par la fonction PUT, à l'aide de la variable FICHIER↑.

Le caractère CTRL-C (c'est-à-dire END OF TEXT) termine l'introduction des données. La variable EOF (END OF FILE) devient alors TRUE. Les données sont ensuite transférées sur la disquette, le fichier étant refermé par CLOSE(<NOM>, LOCK).

L'instruction LOCK signifie que le fichier reste disponible au-delà de l'exécution du programme, en étant effectivement stocké sur la disquette.

Le programme 7.3.2 LIRE a maintenant le rôle de lire ces enregistrements sur la disquette. Les fonctions nécessaires pour lire sur une disquette sont

RESET (<NOM>,' <NOM DU FICHIER>:
<ELEMENT>') ;
et GET (<NOM>).

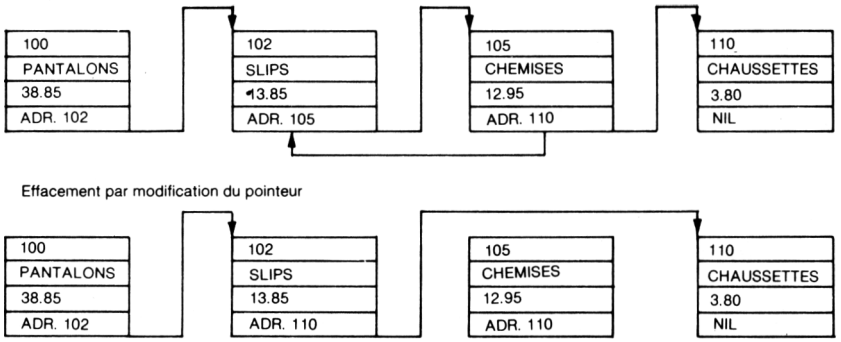


Fig. 7.3. – Effacement d'un enregistrement.

RESET prépare un fichier à la lecture. Le pointeur <NOM>↑ pointe sur le premier enregistrement. Celui-ci, dans notre exemple, est lu par l'effet de l'instruction FICHIER ↑.SORT, et affiché sur l'écran par la procédure SORTAFF.

GET (FICHIER) transmet le pointeur sur les enregistrements successifs, jusqu'à atteindre EOF (FICHIER).

Ces deux exemples se laissent facilement transformer en un véritable répertoire téléphonique, si l'on y ajoute les sous-programmes de recherche de noms ou de numéros.

```
PROGRAM TELEPHONE;
TYPE MOT=ARRAY[1..20]OF CHAR;
     ENREGIST=RECORD
                NOM:MOT;
                PRENOM:MOT;
                NUMERO:MOT;
                END;
VAR N,M:MOT;
    CH:CHAR;
    FICHIER:FILE OF ENREGIST;
    ENRT:ENREGIST;

PROCEDURE ENTREE;
VAR CH:CHAR;
PROCEDURE LIRE(VAR PHR:MOT);
    CONST VIDE=' ';
    VAR I:INTEGER;
    BEGIN
      FOR I:=1 TO 20 DO
      PHR[I]:=VIDE;
      I:=1;
      REPEAT
      READ(PHR[I]);I:=I+1;
      UNTIL EOLN OR (I=20);
    END;

BEGIN
  WITH ENRT DO
```

```
  BEGIN
    WRITE('NOM: ');
    LIRE(NOM);
    WRITE('PRENOM: ');
    LIRE(PRENOM);
    WRITE('NUMERO: ');
    LIRE(NUMERO);
  END;
END;     (*ENTREE*)
BEGIN
WRITELN('METTRE LA DISQUETTE TELE');
WRITE('FAIRE RETURN');READLN;
REWRITE(FICHIER,'TELE:PHONE');
  REPEAT
    ENTREE;
    FICHIER]:=ENRT;
    PUT(FICHIER);
    WRITE('ENCORE UN ENREGISTREMENT (O/N)');
    READ(CH);WRITELN;
  UNTIL CH='N';
  CLOSE(FICHIER,LOCK)
END.
```

Programme 7.3.1 : TELEPHONE

```
PROGRAM LIRE;

TYPE MOT=ARRAY[1..20]OF CHAR;
     ENREGIST=RECORD
                NOM:MOT;
                PRENOM:MOT;
                NUMERO:MOT;
                END;
VAR N,M:MOT;
    CH:CHAR;
    FICHIER:FILE OF ENREGIST;
    SORT:ENREGIST;
PROCEDURE SORTAFF;
```

```
PROCEDURE ECRIT(PHR:MOT);
VAR I:INTEGER;
BEGIN
  FOR I:=1 TO 20 DO WRITE(PHR[I]);
  WRITELN;
END;
BEGIN
  WITH SORT DO
  BEGIN
    ECRIT(NOM);
    ECRIT(PRENOM);
    ECRIT(NUMERO);
  END;
  WRITELN;
END;  (*SORTAFF*)
BEGIN
WRITELN('METTRE LA DISQUETTE TELE');
WRITE('FAIRE RETURN');READLN;
RESET(FICHIER,'TELE:PHONE');
    WHILE NOT EOF(FICHIER)DO
    BEGIN
      SORT:=FICHIER^;
      SORTAFF;
      READLN;
      GET(FICHIER);
    END;
  CLOSE(FICHIER,LOCK);
END
```

Programme 7.3.2 : LIRE

8

Intégration de programmes machine aux programmes en PASCAL

8.1 Un exemple : le traitement de données résultant d'une mesure

Ce chapitre se réfère à un système bien précis : l'Apple II muni du PASCAL UCSD. Les exemples décrits ici s'adapteront cependant facilement à d'autres systèmes de calcul. Le but est de montrer comment l'on insère des parties de programme, écrites en langage machine, à un programme PASCAL. Ce type de programmation sera nécessaire chaque fois qu'il y aura traitement informatique de données provenant du monde extérieur, c'est-à-dire de l'environnement de l'ordinateur. C'est le cas en particulier du traitement des données provenant d'un appareil de mesure. Le facteur temps joue un rôle critique dans leur enregistrement, alors que ce n'est pas le cas pour leur stockage et leur analyse. En revanche, ce type de programmation, nécessitant le langage machine, est relativement complexe. Dans la très grande majorité des cas, le BASIC et, a fortiori, le PASCAL, seront trop lents.

Un autre exemple serait le contrôle de moteurs « pas à pas ». Les impulsions commandant un pas vers la droite ou vers la gauche seront, ici aussi, produites par des parties de programme écrites en langage machine ; il en serait de même pour un programme de production d'impulsions d'horloge. Par contre, la programmation des processus de démarrage ou de freinage, par exemple, pourra s'effectuer en un langage évolué, soit, ici, en PASCAL.

Le programme d'analyse de données mesurées, reproduit ci-dessous à titre d'exemple, est une petite partie d'un programme d'ores et déjà expérimenté en pratique, et qui n'est rendu ici que très partiellement pour des raisons d'encombrement.

Un convertisseur analogique-digital ADC 1210, d'une résolution de 12 bits, branché sur un circuit d'entrée-sortie 6522, transmet les données mesurées au calculateur.

Les programmes en langage machine seront écrits en premier. Nous avons besoin de deux sous-programmes : l'un pour l'initialisation des entrées et des sorties, et l'autre pour l'enregistrement des valeurs mesurées. Dans notre cas quelque peu simplifié, il s'agira d'enregistrer une valeur chaque fois que l'on appuiera sur la touche RETURN (programmes 8.1.1 et 8.1.2).

Le programme 8.1.1 contrôle les entrées et les sorties, ainsi que la mesure. Le bit 5 du port B signale la fin du cycle de conversion, le bit 4 donne le départ de la conversion. Les bits 0 à 7 du port A, et 0 à 3 du port B se chargent de la valeur mesurée de douze bits. La carte comportant le convertisseur et le circuit 6522 est insérée dans le connecteur numéro quatre de l'Apple. La procédure MESURER, écrite en langage machine, voit son nom suivi du nombre **Un**, signifiant le nombre de données transmises au programme PASCAL. Les douze bits provenant du convertisseur ne nécessitent qu'un seul mot, étant donné que le code interne du PASCAL fonctionne en seize bits. Le transfert de la valeur numérique s'effectue par la pile, l'ordre de transfert étant d'abord le LSB (Least Significant Byte), puis le MSB (Most Significant Byte) (1). C'est là que la valeur numérique sera reprise par le programme PASCAL (programme 8.1.2).

```
.MACRO POP
PLA
STA %1
PLA
STA %1+1
.ENDM

.MACRO PUL
```

(1) N.d.t.
LSB = octet de poids faible
MSB = octet de poids fort.

```
            LDA %1+1
            PHA
            STA %1
            PHA
            .ENDM
PORTA       .EQU 0C0C1
PORTB       .EQU 0C0C0
DDRA        .EQU 0C0C3
DDRB        .EQU 0C0C2
RETURN      .EQU 00
            .PROC INIT
            POP RETURN
            LDA #40
            STA DDRB
            PUL RETURN
            RTS
            .PROC MESURER,1
            POP RETURN
            LDA #10
            STA PORTB
            LDA #00
            STA PORTB
END         LDA PORTB
            AND #40
            BNE IND
            LDA PORTA
            PHA
            LDA PORTB
            AND #0F
            PHA
            PUL RETURN
            RTS
            .END
```

Programme 8.1.1 : Mesure (en langage machine)

Les deux programmes machine ont été écrits en assembleur, pour être rangés sur disquette comme un fichier.

Ces routines sont ensuite déclarées en tant que procédures « externes ».

La compilation du programme PASCAL étant effectuée, il faut réunir ce dernier au programme machine. Cette fonction est assurée par l'éditeur de liens. C'est un programme du système d'exploitation reliant les routines externes, ou traduites à l'avance, au programme principal.

En ce qui concerne le déroulement de l'utilisation de l'éditeur de liens, il est recommandé de se référer aux modes d'emploi des différents ordinateurs.

En guise de conclusion à cette partie de l'ouvrage, consacrée aux instructions, l'on trouvera à la figure 8.1 le diagramme syntaxique du programme PASCAL.

```
PROGRAM MESURE;
CONST I=50;
VAR J:INTEGER;
    VALEUR:INTEGER;
    VALEURS:ARRAY[1..I]OF INTEGER;

PROCEDURE INIT;EXTERNAL;
PROCEDURE MESURE(VAR W:INTEGER);
EXTERNAL;

BEGIN
  WRITELN('DEBUT DE LA MESURE');
  J:=1;
  INIT;READLN;
  REPEAT
    MESURE(VALEUR);
    VALEUR[J]:=VALEUR;
    J:=J+1;
    READLN;
  UNTIL J > I;

  FOR J:=1 TO I DO
  WRITELN(I,'    ',VALEURS[I]);
END.
```

Programme 8.1.2 : Programme de mesure en PASCAL

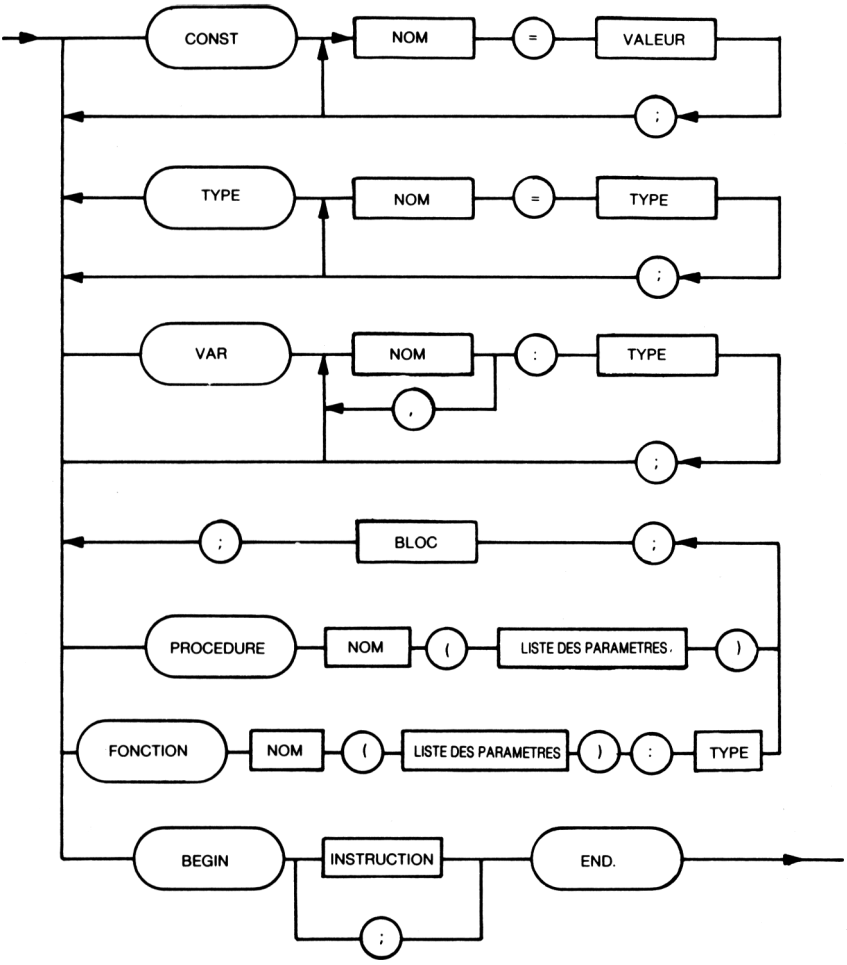


Fig. 8.1. – Diagramme syntaxique d'un programme.

9

Une collection de programmes variés

L'on trouvera maintenant une collection de différents programmes servant d'exemples de solution à des problèmes de programmation. Mais comme tout programmeur a son style propre, les programmes peuvent, dans presque tous les cas, être formulés différemment. Ces programmes serviront donc surtout pour réviser les connaissances en programmation.

Le langage PASCAL ne possède pas d'instruction permettant l'accès direct à une cellule mémoire. La construction (quelque peu aventureuse) d'un RECORD permet d'y remédier. Le programme 9.1.1 PEEK16 prend le contenu de deux cellules mémoire contiguës, l'adresse indiquée étant l'octet de poids faible, et le contenu de l'adresse suivante étant l'octet de poids fort. Le programme 9.1.2 ne charge le contenu que d'une seule cellule.

Le programme 9.1.3 POKE 8 contient un sous-programme POKE dont la fonction est de déposer un octet dans une cellule mémoire déterminée par une adresse décimale. Ce programme contient aussi une procédure PEEK 8, affichant les données enregistrées sur commande de la touche RETURN.

```
PROGRAM PEEK;

VAR L:INTEGER;

FUNCTION PEEK16(LOC:INTEGER):INTEGER;

VAR X:RECORD
      CASE BOOLEAN OF
```

```
        TRUE:(INT:INTEGER);
        FALSE:(PTR:INTEGER)
        END;
BEGIN
  X.INT:=LOC;
  PEEK16:=X.PTR^;
END;

BEGIN
WRITELN('INDIQUER L ADRESSE DECIMALE');
READLN(L);
WRITELN(PEEK16(L));
END.
```

Programme 9. i.1 : PEEK16

```
PROGRAM PEEK;

VAR L:INTEGER;

FUNCTION PEEK8(LOC:INTEGER):INTEGER;
TYPE BYTE=PACKED ARRAY[0..1]OF 0..255;

VAR X:RECORD
        CASE BOOLEAN OF
        TRUE:(INT:INTEGER);
        FALSE:(PTR:^BYTE)
        END;
BEGIN
  X.INT:=LOC;
  PEEK8:=X.PTR^[0];
END;

BEGIN
WRITELN('INDIQUER L ADRESSE DECIMALE');
READLN(L);
WRITELN(PEEK8(L));
END.
```

Programme 9.1.2 : PEEK8

```
PROGRAM POKE8;
VAR L,I:INTEGER;

PROCEDURE POKE(LOC,INH:INTEGER);
TYPE BYTE=PACKED ARRAY[0..1]OF 0..255;

VAR X:RECORD
      CASE BOOLEAN OF
      TRUE:(INT:INTEGER);
      FALSE:(PTR:^BYTE)
      END;
BEGIN
X.INT:=LOC;
X.PTR^[0]:=INH;

END;

FUNCTION PEEK8(LOC:INTEGER):INTEGER;
TYPE BYTE=PACKED ARRAY[0..1]OF 0..255;

VAR X:RECORD
      CASE BOOLEAN OF
      TRUE:(INT:INTEGER);
      FALSE:(PTR:^BYTE);
      END;
BEGIN
  X.INT:=LOC;
  PEEK8:=X.PTR^[0];
END;

BEGIN
WRITELN('INDIQUER L ADRESSE DECIMALE');
READLN(L);
WRITELN('INDIQUER LE CONTENU ');
READLN(I);
POKE(L,I);
READLN;
WRITELN(PEEK8(L));
END.
```

Programme 9.1.3 : POKE8

9.2 Méthode de Horner

Le programme 9.2.1 HORNER calcule la valeur du polynôme (dont le degré doit être inférieur à 20) d'après la méthode de Horner. L'on introduira les coefficients du polynôme, et la valeur X de l'abscisse pour laquelle la fonction doit être calculée.

```
PROGRAM HORNER;

VAR X,Y,Z:REAL;
    I,N:INTEGER;
    A:ARRAY[0..20]OF REAL;

BEGIN
  WRITE('DEGRES DU POLYNOME ');READLN(N);
  WRITELN('INDIQUER LES COEFFICIENTS ');
  WRITELN('EN COMMENCANT PAR A[N]');WRITELN;
  FOR I:=N DOWNTO 0 DO
  BEGIN
    WRITE('A[',I,']= ');READLN(A[I]);
  END;
  WRITELN;
  WRITE('X=');READLN(X);
  Y:=0;
  FOR I:=N DOWNTO 0 DO Y:=Y*X+A[I];
  WRITELN('Y=',Y:5:2);
END.
```

Programme 9.2.1 : HORNER

9.3 Multiplication des polynômes

Le programme 9.3.1 MUL multiplie entre eux deux polynômes de degré inférieur à dix, et inscrit le résultat dans le tableau numérique E.

Les coefficients sont stockés dans un tableau de la manière suivante : la valeur d'index 0 sera le coefficient de X^0, la valeur d'index 1 sera le coefficient de X^1, etc. Cette disposition facilite ensuite la multiplication des polynômes (PROCEDURE POLYMUL).

Les autres procédures sont destinées aux entrées-sorties. Nous les retrouverons dans les programmes ADD et DER.

```
PROGRAM MUL;

CONST G=20;

TYPE POLY=ARRAY[0..G]OF REAL;

VAR P1,P2,E:POLY;
    I,J,K:INTEGER;

PROCEDURE POLYENT(VAR P:POLY;VAR K:INTEGER);

VAR R:REAL;
    I:INTEGER;
BEGIN
  WRITE('DEGRE DU POLYNOME ');READLN(K);
  WRITELN('INDIQUER LES COEFFICIENTS');
  WRITELN('LE PREMIER ETANT P[',K,']');
  FOR I:=K DOWNTO 0 DO
  BEGIN
    WRITE('P[',I,']=');READLN(P[I]);
  END;
END;  (*POLYENT*)


PROCEDURE POLYSORT(P:POLY;K:INTEGER);
VAR I:INTEGER;
BEGIN
  WRITELN('LES COEFFICIENTS DU POLYNOME');
  WRITELN;
  FOR I:=0 TO K DO
  WRITELN('P[',I,']=',P[I]);
END;  (*POLYSORT*)

PROCEDURE POLYMUL(N,M:INTEGER;P1,P2:POLY;
                  VAR L:INTEGER;VAR E:POLY);
VAR J,I:INTEGER;

BEGIN
FOR I:=0 TO G DO E[I]:=0;
```

```
 FOR J:=0 TO N DO
  FOR I:=0 TO M DO
  E[I+J]:=E[I+J]+P1[J]*P2[I];
  L:=N+M;
 END;
BEGIN  (*PR PRINCIPAL*)

WRITELN('1.POLYNOME: ');
POLYENT(P1,I);
WRITELN('2.POLYNOME: ');
POLYENT(P2,J);
POLYMUL(I,J,P1,P2,K,E);
POLYSORT(E,K);

END.
```

Programme 9.3.1 : MUL

9.4 Addition de polynômes

La procédure POLYADD additionne deux polynômes P1 et P2.

```
PROGRAM ADD;

CONST G=20;

TYPE POLY=ARRAY[0..G]OF REAL;

VAR P1,P2,E:POLY;
    I,J,K:INTEGER;

PROCEDURE ZERO(VAR P:POLY);
VAR I:INTEGER;
BEGIN
  FOR I:=0 TO G DO P[I]:=0;
END;  (*ZERO*)

PROCEDURE POLYENT(VAR P:POLY;VAR K:INTEGER);
VAR R:REAL;
    I:INTEGER;
```

```
BEGIN
    ZERO(P);
    WRITE('DEGRE DU POLYNOME ');READLN(K);
    WRITELN('INDIQUER LES COEFFICIENTS');
    WRITELN('EN COMMENCANT PAR P[',K,']');
    FOR I:=K DOWNTO 0 DO
    BEGIN
      WRITE('P[',I,']=');READLN(P[I]);
    END;
END;  (*POLYENT*)

PROCEDURE POLYSORT(P:POLY;K:INTEGER);
VAR I:INTEGER;
BEGIN
  WRITELN('LES COEFFICIENTS DU POLYNOME');
  WRITELN;
  FOR I:=0 TO K DO
  WRITELN('P[',I,']=',P[I]);
END;  (*POLYSORT*)

PROCEDURE POLYADD(N,M:INTEGER;P1,P2:POLY;
                  VAR L:INTEGER;VAR E:POLY);
VAR J,I:INTEGER;

BEGIN
IF N >=M THEN L:=N;
ZERO(E);
FOR I:=0 TO L DO E[I]:=P1[I]+P2[I];
END;  (*POLYADD*)

BEGIN  (*PR PRINCIPAL*)

WRITELN('1.POLYNOME: ');
POLYENT(P1,I);
WRITELN('2.POLYNOME: ');
POLYENT(P2,J);
POLYADD(I,J,P1,P2,K,E);
POLYSORT(E,K);
END.
```

Programme 9.4.1 : ADD

9.5 Dérivée d'un polynôme

Les règles de calcul de la dérivée d'un polynôme sont ici appliquées dans la procédure POLYDER. Seuls le programme principal et la procédure de ce programme sont reproduits ici. Le reste du programme correspond à 9.4.1 ou 9.5.1.

```
PROCEDURE POLYDER(N:INTEGER;P1:POLY;
                  VAR L:INTEGER;VAR E:POLY);

VAR J,I:INTEGER;

BEGIN
ZERO(E);
FOR I:=1 TO N DO
E[I-1]:=P1[I]*I;
L:=N-1;
END;  (*POLYDER*)

BEGIN (*PR PRINCIPAL*)

WRITELN('POLYNOME: ');
POLYENT(P1,I);
POLYDER(I,P1,K,E);
POLYSORT(E,K);

END
```

Programme 9.5.1 : POLYDER

9.6 Analyse de Fourier

Ce programme exécute une analyse de Fourier sur des valeurs de fonction déterminées par l'utilisateur. Le procédé mathématique employé est quelque peu différent des méthodes habituelles ; l'on en trouvera une description complète dans l'ouvrage « Méthodes mathématiques pour calculateurs digitaux », de A. Ralston et H.S. Wilf, éditions Oldenburg Verlag.

Les données nécessaires sont les valeurs NP de la fonction, et le nombre NH d'harmoniques à calculer, avec NP>2*NH+1. Les sinus et cosinus sont affichés.

```
PROGRAM ANALYSEFOURIER;
USES TRANSCEND;
CONST PI=3.1415926;N=33;

TYPE VALFONC=ARRAY[1..N]OF REAL;

VAR F:VALFONC;
    DY:REAL;
    NP,NH,I:INTEGER;

PROCEDURE LIRE(VAR NH,NP:INTEGER);
BEGIN
  REPEAT WRITE(' INDIQUER NH ET NP  ');READLN(NH,NP)
  UNTIL NP > (2*NH+1);
END;  (*LIRE*)

PROCEDURE FOURIER(NH,NP:INTEGER;F:VALFONC);

VAR Z,S1,C1,C,S,Q,U0,U1,U2,A,B:REAL;
    KLN,P:INTEGER;

BEGIN
  WRITELN;
  WRITELN('  N   COS          SIN');
  WRITELN(' ----------------------');
  Z:=2/NP;C1:=COS(Z*PI);S1:=SIN(Z*PI);
  C:=1;S:=0;P:=0;
  REPEAT
    WRITELN;
    U1:=0;U2:=0;
    KLN:=NP-1;
    REPEAT
      U0:=F[KLN]+2*C*U1-U2;
      U2:=U1;U1:=U0;KLN:=KLN-1;
    UNTIL KLN=0;
    A:=Z*(F[0]+C*U1-U2);
    B:=Z*S*U1;
    IF P=0 THEN A:=A/2;
    WRITELN(P:3,A:10:5,B:10:5);
    BEGIN
      Q:=C1*C-S1*S;
      S:=C1*S+S1*C;
      C:=Q;P:=P+1;
    END;
  UNTIL P=NH+1;
END;   (*FOURIER*)
```

```
BEGIN  (*PR PRINCIPAL*)
LIRE(NH,NP);
WRITELN;WRITELN;
WRITELN('EXEMPLE DENT DE SCIE');
WRITELN;
F[0]:=1;DY:=2/NP;
FOR I:=1 TO NP-1 DO
F[I]:=F[I-1]+DY;
F[0]:=0;

FOURIER(NH,NP,F);

END.
```

Programme 9.6.1 : ANALYSEFOURIER

9.7 Comparaison entre deux lettres

Le programme ZCOMP compare deux lettres données par l'utilisateur, et détermine leur ordre dans l'alphabet.

```
PROGRAM ZCOMP;
CONST VIDE=' ';
VAR S:INTEGER;
    Y,X:CHAR;
BEGIN
  WRITELN('DONNEES');
  WRITELN('=======');
  READ(X,Y);
  IF X=Y THEN S:=0;
  IF(X > Y)OR(Y=VIDE)THEN S:=-1;
  IF(X < Y)OR(Y=VIDE)THEN S:=1;
  WRITELN('CAS ',S);
  IF S=0 THEN WRITELN('LES DEUX LETTRES SONT IDENTIQUES');
  IF S=1 THEN WRITELN(X,' EST AVANT ',Y);
  IF S=-1 THEN WRITELN(Y,' EST AVANT ',X);
END.
```

Programme 9.7.1 : ZCOMP

9.8 Programme SPLIT

Le programme SPLIT, en présence d'un nombre à quatre chiffres, le dissocie en ses chiffres constitutifs et les place dans un tableau.

```
PROGRAM SPLIT;
  CONST N=4;
  TYPE TAB=ARRAY[1..N]OF INTEGER;
  VAR I,C:INTEGER;
      A:TAB;
BEGIN
  WRITELN('DONNEES');
  WRITELN('=======');
  READ(C);
  A[1]:=C DIV 1000;
  A[2]:=(C MOD 1000)DIV 100;
  A[3]:=(C MOD 100)DIV 10;
  A[4]:=C MOD 10;
  FOR I:=N DOWNTO 1 DO WRITE(A[I],' ')
END;
```

Programme 9.8.1 : SPLIT

9.9 Programme QUICK

Le programme suivant est une seconde version de QUICKSORT. L'on emploie ici non pas une méthode récursive, mais une zone mémoire organisée en pile.

```
PROGRAM QUICK;
(*LE PROGRAMME LIT N NOMBRES,LES CLASSE
PUIS AFFICHE LE RESULTAT*)
CONST N=8;
TYPE INDEX=0..N;
     ZSUITE=ARRAY[INDEX]OF INTEGER;
VAR I:INDEX;
    A:ZSUITE;
    P:INTERACTIVE;

PROCEDURE QUICKSORT(VAR A:ZSUITE);
(*CLASSE LES NOMBRES SANS FAIRE APPEL
A LA RECURSIVITE.
ON UTILISE UNE PILE DANS LAQUELLE SONT
STOQUES LES INDEX DROIT ET GAUCHE DE CE QUI
RESTE A CLASSER.*)
```

```
CONST M=12;
VAR I,J,L,R:INDEX;
    X,W:INTEGER;
    S:0..M;
    STACK:ARRAY[1..M]OF RECORD L,R:INDEX
    END;

BEGIN S:=1;STACK[1].L:=1;STACK[1].R:=N;
  REPEAT
    L:=STACK[S].L;R:=STACK[S].R;S:=S-1;
  REPEAT
    I:=L;  J:=R;
           X:A[(L+R) DIV 2];
    REPEAT
      WHILE A[I] < X DO I:=I+1;
      WHILE A[J] > X DO J:=J-1;
      IF I<=J THEN
      BEGIN W:=A[I];A[I]:=A[J];A[J]:=W;
            I:=I+1; J:=J-1;
      END;
    UNTIL I > J;
    IF I < R THEN
    BEGIN
      S:=S+1;STACK[S].L:=I;STACK[S].R:=R;
    END;
    R:=J;
    UNTIL L>= R;
  UNTIL S=0;
  END;(*QUICKSORT*)

  BEGIN (*PR PRINCIPAL*)
  WRITELN;
  WRITELN('NOMBRES NON CLASSES:');
  WRITELN('--------------------');
    FOR I:=1 TO N DO
    BEGIN READ(A[I]);WRITE(' ');END;
  QUICKSORT(A);
  WRITELN;WRITELN;
```

```
  WRITELN('NOMBRES CLASSES:');
  WRITELN('----------------');
  FOR I:=1 TO N DO
    WRITE(A[I],' ');
  WRITELN;

END.
```

Programme 9.9.1 : QUICK

10

Comment élaborer un programme en PASCAL

10.1 Exemple de programmation sur APPLE II

Sont nécessaires, en dehors de l'APPLE II, la carte langage et deux lecteurs de disquettes. Les disquettes APPLE 1 et APPLE 2 seront respectivement placées dans les lecteurs 1 (4 en PASCAL) et 2 (5 en PASCAL). Dès la mise en route de l'ordinateur, il apparaît sur l'écran une ligne d'instructions de choix des programmes :

E(DIT, R(UN, F(ILE, C(OMP, L(INK, X(ECUTE, A(SSEM, D(EBUG? [II. 1]

La dernière indication représente le numéro de la version de l'ensemble PASCAL utilisé.

L'on appellera d'abord le F(ILE, pour y donner la date par D(ATE. Cette date sera reprise par le directory, lors de l'enregistrement d'un programme. Ceci permet de conserver une indication sur la date d'écriture des programmes.

L'écriture d'un programme commence par N(EW, c'est-à-dire la mise à disposition d'un fichier de travail. S'il subsiste, à ce moment précis, un ancien fichier, le programme demande l'autorisation de l'effacer. Si la réponse est Y, le fichier est effacé. L'on peut alors quitter le Filer par Q(UIT et rejoindre l'éditeur de texte par E(DITOR.

Avant d'accéder à l'éditeur, le programme demande si l'on ne désire pas conserver un fichier. La touche RETURN permet enfin de rejoindre l'éditeur. L'on peut désormais introduire le texte par I(NSERT. L'introduction sera conclue par CTRL-C (presser à la fois

Control et C). CTRL-C signifie ETX (End of Text) en code ASCII. Il est ensuite possible de corriger le texte, le curseur étant dirigé par les touches indiquées à la figure 10.1 :

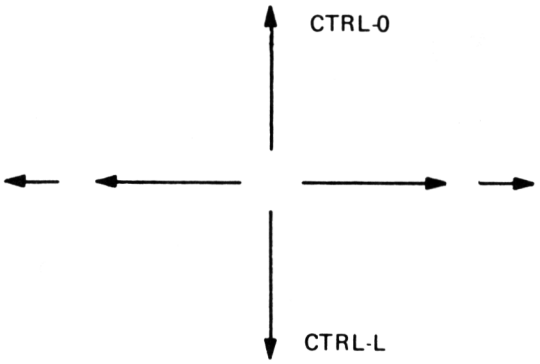


Fig. 10.1. – Mouvements du curseur.

P permet de sauter une page entière, la direction étant donnée par le caractère placé devant le EDIT de la ligne des instructions. <signifie « aller vers la fin » et > « aller vers le début ».

Les instructions de correction sont D(ELETE, I(INSERT, et X(CHANGE. Toute correction doit être suivie d'un CTRL-C.

Toutes les corrections ayant été faites, l'on quittera l'éditeur par Q(UIT.

Le programme continue par une requête où, grâce à U(PDATING, l'on pourra transférer le fichier de travail du buffer d'entrée sur la mémoire utilisatrice. Ce n'est qu'à partir de là que ce fichier est enregistré sur la disquette APPLE 1, à la disposition d'un traitement ultérieur.

Il est recommandé, lors de l'écriture de longs programmes, de fréquemment « sauver » ainsi le travail déjà accompli, de manière à ce que tout ne soit pas perdu si le système se « plante ».

C(OMPILE traduit alors le programme. En présence d'une erreur de syntaxe, la traduction est stoppée ; il est possible, dans ce cas, de se faire expliquer l'erreur, de la corriger ou de poursuivre la compilation.

Lorsque l'on a obtenu une compilation exempte d'erreurs, on peut passer à son exécution par R(UN. Bien sûr, il peut encore y avoir des erreurs de temps d'exécution ou des erreurs logiques. Si

l'on désire copier le programme du fichier de travail sur un autre fichier, on pourra le faire par S(AVE à partir du Filer. L'on indiquera, dans ce cas, le nom du fichier et celui du programme, sous la forme :

<NOM DU FICHIER>:<NOM DU PROGRAMME>

Pour cette raison, le nom du fichier est identique au nom de la disquette sur laquelle est stocké le programme. Il est conseillé d'adopter ce type de dénomination, qui permet d'être sûr de ne pas enregistrer des programmes sur des mauvaises disquettes (disquettes du système).

10.2 Quelques particularités de l'APPLE

Le PASCAL UCSD employé sur l'APPLE présente quelques différences par rapport au PASCAL standard. Certaines, très utiles en programmation, sont explicitées ci-dessous.

10.2.1 Affichage de résultats par l'imprimante

Le système d'exploitation gère les périphériques comme s'ils étaient des fichiers. PRINTER: et CONSOLE: sont des noms de fichiers comparables aux noms des disquettes. Pour faciliter le transfert des données entre l'ordinateur et les périphériques, il a été introduit un type de variable particulier : le type INTERACTIVE. Si, par exemple, on désire un affichage sur l'imprimante, on déclarera, dans la section de déclaration, P:INTERACTIVE ;. On ouvre le fichier « imprimante » par RESET(P,'PRINTER:') ; et tous les textes à imprimer le seront par WRITELN(P, <VARIABLE>). CLOSE(P) terminera l'affichage par l'imprimante.

10.2.2 La procédure SEEK

Les enregistrements, sur un fichier de données, sont toujours stockés séquentiellement, les uns après les autres. Pour pouvoir accéder sélectivement à un certain enregistrement, on utilise la procédure

SEEK(<NOM DU FICHIER>, <NOMBRE ENTIER>)

Cette fonction contrôle le pointeur de données de telle façon que la prochaine fonction PUT ou GET soit appliquée à l'enregistrement déterminé par <NOMBRE ENTIER>.

Programme de démonstration 10.2.2.1 SEEKDEMO

Un tableau numérique est stocké sur le fichier TEST:TEST, puis à nouveau recherché, affecté à un autre tableau numérique, et enfin affiché.

```
PROGRAM SEEKDEMO;

TYPE TAB=ARRAY[1..10]OF INTEGER;
VAR A,B:TAB;
      I:INTEGER;
      F:FILE OF TAB;

BEGIN
  WRITELN('INTRODUIRE LA DISQUETTE TEST:');
  WRITE('PUIS FAIRE RETURN');READLN;
  REWRITE(F,'TEST:TEST');
  FOR I:=1 TO 10 DO A[I]:=11-I;
  F^:=A;PUT(F);
  FOR I:=1 TO 10 DO A[I]:=I;
  F^:=A;PUT(F);
  SEEK(F,0);GET(F);
  B:=F^;
  FOR I:=1 TO 10 DO WRITE(B[I],'..');
END.
```

Programme 10.2.2.1 : SEEKDEMO

10.2.3 La procédure PAGE

Cette procédure envoie un signal TOP OF FORM (12 en ASCII) vers un fichier déterminé.

PAGE(<NOM>) ;

L'instruction PAGE(OUTPUT) éteint l'écran et place le curseur en haut et à gauche.

10.2.4 La procédure GOTOXY

Cette procédure permet de diriger le curseur sur l'écran. L'on appelle cette fonction par

GOTO(<COORDONNEE X>, <COORDONNEE Y>) ;

L'angle supérieur gauche reçoit les coordonnées 0,0 alors que l'inférieur droit reçoit les coordonnées 79,23.

Le programme 10.2.4.1 FICHE détermine le format des fiches d'un fichier, de telle manière que l'on puisse y inscrire des adresses. Il n'y a pas d'autre traitement des données enregistrées. Ce programme peut cependant être utilisé, sous une forme modifiée, avec les programmes 7.2.1 LISTEPRIX ou 7.3.1 TELEPHONE.

L'usage de la fonction GOTOXY est restreint à l'écran APPLE. Les autres appareils obligent à une modification du système d'exploitation.

Les autres fonctions sont identiques à celles des versions UCSD de l'ordinateur OHIO, et seront traitées au chapitre suivant.

```
PROGRAM FICHE;
CONST VIDE='                              ';
TYPE PHR=ARRAY[1..20]OF CHAR;
VAR ENT:PHR;
    I:INTEGER;
PROCEDURE LIRE;
VAR I:INTEGER;
BEGIN
  FOR I:=1 TO 20 DO ENT[I]:=' ';
  I:=1;
  REPEAT
    READ(ENT[I]);I:=I+1;
  UNTIL EOLN OR(I=20);
END;

BEGIN
PAGE(OUTPUT);
GOTOXY(1,5);
FOR I:=1 TO 31 DO WRITE('*');
GOTOXY(1,6);WRITE('*NOM     :',VIDE,'*');
GOTOXY(1,7);WRITE('*PRENOM  :',VIDE,'*');
GOTOXY(1,8);WRITE('*RUE     :',VIDE,'*');
GOTOXY(1,9);WRITE('*VILLE   :',VIDE,'*');
GOTOXY(1,10);WRITE('*TEL      :',VIDE,'*');
GOTOXY(31,10);WRITE('*');
GOTOXY(1,11);FOR I:=1 TO 31 DO WRITE ('*');
```

```
GOTOXY(10,6);LIRE;
GOTOXY(10,7);LIRE;
GOTOXY(10,8);LIRE;
GOTOXY(10,9);LIRE;
GOTOXY(18,10);LIRE;
END.
```

```
*********************************
*NOM     :                      *
*PRENOM  :                      *
*RUE     :                      *
*VILLE   :                      *
*           TEL:                *
*********************************
```

Programme 10.2.4.1 : FICHE

11

PASCAL UCSD sur le C4PMF d'Ohio Scientific

Parmi ce que l'on appelle les langages évolués, le PASCAL est celui qui connaît la plus grande expansion actuellement. Ce langage, dénommé d'après le philosophe et mathématicien français Blaise Pascal (1623-1662), fut développé par le professeur Nicolas Wirth (de l'Université Technique de Zurich) au début des années soixante-dix. Le premier compilateur PASCAL fut alors développé par Urs Aumann, en collaboration avec le professeur Wirth.

Le PASCAL du professeur Wirth est considéré comme le PASCAL standard. Le PASCAL UCSD (University of California at San Diego) représente une évolution considérable par rapport à celui-ci.

Le langage de programmation PASCAL est apparenté à l'ALGOL 60, tous deux étant, à l'inverse de BASIC, des langages structurés.

Tout programme PASCAL est construit de la façon suivante :

En-tête
Section de déclaration Bloc
Programme principal

Le PASCAL donne d'autre part la possibilité d'écrire ses propres procédures (un peu comme les subroutines en BASIC). Une procédure est construite comme un bloc. Tout bloc contient une section de déclaration et un programme principal. Il est possible de réunir plusieurs blocs, ou de les combiner entre eux. Ceci mène à l'existence de plusieurs types de variables (puisque chaque bloc possède

une section de déclaration) : celles que l'on peut utiliser partout dans le programme, appelées « variables globales », et celles qui ne sont définies qu'à l'intérieur d'un bloc, les « variables locales ».

PASCAL UCSD sur le C4PMF d'Ohio Scientific

La société SOFTECH MICROSYSTEMS livre le PASCAL UCSD (ainsi que le FORTRAN) sur sept mini-disquettes :

PASCAL 1
PASCAL 2
PASCAL 3
UTIL 1
UTIL 2
UTIL 3
UTIL 4

Le système est initialisé par la disquette PASCAL 1 (« boot »). Mais, comme le montre le contenu de PASCAL 1, l'utilisateur ne dispose que d'un seul bloc pour stocker un fichier personnel. Comme quatre blocs, au minimum, sont nécessaires, il faut aménager de l'espace en transférant des fichiers sur une autre disquette, par un second lecteur de disquettes (le lecteur numéro 1 s'appelle, en PASCAL UCSD, le Nr 4, le second lecteur étant le Nr 5).

Nous verrons plus tard comment opérer.

Tout d'abord, une vue d'ensemble sur les instructions (ce sont des commandes en PASCAL UCSD) ; la structure du PASCAL UCSD prévoit plusieurs niveaux de commande. Au début, on se trouve toujours au niveau le plus superficiel. Chaque commande fait passer à un niveau plus bas. A l'inverse, l'instruction 'Q' pour Quit fait repasser à un niveau plus extérieur.

Les commandes se contrôlent en tapant une seule lettre. Parfois, une ligne ne suffit pas à visualiser toutes les commandes possibles. Il suffit alors de frapper la touche '?' pour voir celles qui restent.

Résumé des commandes du premier niveau

Il devrait y avoir, affiché sur l'écran :

Command : E(dit, R(un, F(ile, C(omp, L(ink, X(ecute,?

1. E(dit

L'éditeur sert à écrire un nouveau programme ou à modifier un programme existant. Au cas où il existe un programme utilisateur sur la disquette système (sous le nom SYSTEM.WRK.TEXT), celui-ci est chargé, puis affiché, et peut être modifié.

2. R(un

La commande Run permet l'exécution d'un programme déjà compilé et stocké sous le nom de SYSTEM.WRK.CODE. En l'absence d'un fichier de ce nom, il y a automatiquement appel du compilateur-système et compilation du fichier SYSTEM.WRK.TEXT, puis exécution de celui-ci (si la compilation a été possible).

3. F(iler

Ses possibilités principales sont la copie de fichiers d'une disquette sur une autre, et la préparation des disquettes vides.

4. C(omp

Le compilateur du système sert à traduire le fichier de travail (SYSTEM.WRK.TEXT) en code machine. Il est impossible de traduire un fichier-source (= fichier-texte) entaché d'erreurs de syntaxe. Lorsque le compilateur détecte une erreur de syntaxe, celle-ci est affichée, et l'on peut directement accéder à l'éditeur pour corriger l'erreur.

Une fois traduit de façon satisfaisante, le code machine est stocké sous le nom SYSTEM.WRK.CODE sur la disquette-système.

5. L(ink

Le linker du système permet d'intégrer des routines déjà traduites, au fichier de travail.

6. X(ecute

Cette commande est proche de RUN. Le système demande le nom du programme à exécuter. Ce programme doit déjà être traduit, c'est-à-dire être stocké en NOM.CODE.

Répartition des fichiers du PASCAL C4PMF

PA4 : Drive # 4 : nombres entiers seulement		
SYSTEM. PASCAL	40	6
SYSTEM. MISCINFO	1	46
SYSTEM. INTERP	20	47
SYSTEM. BIOS	7	67
SYSTEM. FILER	28	74
SYSTEM. LIBRARY	19	102
*	31	121

PA4MFP : Drive # 4 : arithmétique à virgule flottante		
SYSTEM. PASCAL	40	6
SYSTEM. MISCINFO	1	46
SYSTEM. INTERP	23	47
SYSTEM. BIOS	7	70
SYSTEM. FILER	28	77
SYSTEM LIBRARY	19	105
*	28	124

PA5 : DRIVE # 5

SYSTEM. EDITOR	45	6
SYSTEM. COMPILER	68	51
SYSTEM. LINKER	22	119
*	11	141

Contenu des disquettes : (respectivement nom, longueur en blocs, bloc de départ)

PASCAL 1 :

SYSTEM. PASCAL	40	6
SYSTEM. MISCINFO	1	46
SYSTEM. INTERP	20	47
SYSTEM. BIOS	7	67
SYSTEM. SWAPDISK	4	74
SYSTEM. FILER	28	78
SYSTEM. EDITOR	45	106
*	1	151

PASCAL 2 :

SYSTEM. COMPILER	68	6
SYSTEM. SYNTAX	14	74
SYSTEM. ASSMBLER	49	88
6500. OPCODES	2	137
ASM. ERRORS	7	139
6502. INFO. TEXT	4	146
*	2	150

PASCAL 3 :

SYSTEM. LINKER	22	6
SYSTEM. LIBRARY	19	28
FORTRAN. CODE	89	47
*	16	136

UTIL 1 :

REC. DOCUM. TEXT	6	6
RECOVER. G. CODE	8	12
SAMPLEGOTO. TEXT	4	20
YALOE. CODE	14	24
LIBMAP. CODE	7	38
MARKDUPDIR. CODE	4	45
COPYDUPDIR. CODE	3	49
LIBRARY. CODE	7	52
SCREENTEST. CODE	19	59
BINDER. CODE	6	78
DISASM. II. CODE	41	84
OPCODES. II. O	5	125
SC. GEN. CODE	16	130
*	6	146

UTIL 2 :

PATCH. DOC. TEXT	24	6
PATCH. CODE	34	30
FLIPCODE. CODE	8	64
OPFILE	1	72
FLIPDIR. CODE	3	73
RELOC. CODE	7	76
SQUISH. CODE	8	83
BIOS. CODE	8	91
INTCON. CODE	4	99
EXTCON. CODE	3	103
EMPTYUSR. CODE	3	106
EMPTYREM. CODE	3	109
EMPTYRN. CODE	3	112
FP. INTERP.	23	115
*	14	138

UTIL 3:

II. SYSTEM	33	6
SETUP. CODE	24	39
STARTUP. CODE	3	63
BIOSUTIL. CODE	7	66
DISKUTIL. CODE	16	73
RTUNIT. CODE	29	89
RELOC. DOC. TEXT	12	118
FLIP. DOC. TEXT	16	130
*	6	146

UTIL 4 :

ORIENTER. CODE	12	6
EDITDEMO. TEXT	4	18
COMPDEMO. TEXT	6	22
NAMEFILE	3	28
CORYSCUNIT. CODE	3	31
SCDEMo. CODE	6	34
READ. ME. TEXT	4	40
UPDATE. TEXT	8	44
UPDATE. CODE	4	52
*	96	56

* disponible pour un fichier de travail.

Réalisation des fichiers ci-dessus

Trois mini-disquettes (5 1/4'') vierges sont nécessaires. Ces disquettes doivent d'abord être formatées ; l'on procédera de la manière suivante :

Initialiser le système par la disquette PASCAL 1 dans le lecteur 4. Après apparition du menu des commandes de niveau superficiel, en haut de l'écran, on tape X pour X(ecute. Normalement, il est alors affiché :

Execute what file ?

Nous tapons alors :

UTIL 3:DISKUTIL ou # 5:DISKUTIL

Après avoir introduit la disquette UTIL 3 dans le lecteur # 5, on appuie sur RETURN, ce qui a pour effet de charger le fichier. On devrait normalement lire sur l'écran :

DISK:F(ormat B(ootstrap C(opy U(nit Q(uit

Appuyons sur F ; on lit alors :

Format:W(hole P(artial Q(uit

Appuyons sur W ; on lit maintenant :

Format what unit ?

Nous tapons 5 RETURN, il apparaît la ligne suivante :

Put disk in # 5 and type return.

Attention

Avant d'appuyer sur RETURN, il faut sortir la disquette UTIL 3 du drive 5, et engager une disquette vierge ; celle-ci sera donc formatée lorsque l'on aura appuyé sur RETURN. Une fois ce processus terminé, il apparaît à nouveau la ligne des commandes de formatage. Nous formaterons les deux autres disquettes de la même manière que la première.

Nous avons maintenant besoin du Filer ; il nous faut d'abord quitter le programme DISKUTIL, en pressant deux fois la touche Q, de manière à retrouver le niveau le plus superficiel ; puis, appuyer sur F pour F(iler.

Tapons Z pour Z(ero, pour donner un nom à la disquette et pour initialiser le directory.

La première question sera :

What Vol ?	ce à quoi nous répondrons par # 5
Duplicate dir ?	notre réponse : N

Number of blocks ?	notre réponse : 152
New vol name ?	ici, nous répondrons par PA4 (pour les autres disquettes, ce sera successivement PA4MFP puis PA5)
PA4 correct ?	notre réponse : Y
PA4 zeroed	

PA4 est maintenant prêt à recevoir des fichiers. Nous sommes encore sur le niveau du F(iler et, pour transférer les fichiers, nous emploierons la commande T(ransfert.

La première question sera ici :

Transfer what file ?	et nous répondrons par PASCAL 1: SYSTEM.PASCAL (RETURN) ; Le système, après avoir trouvé le fichier en question, demande :
To where ?	notre réponse : PA4:$ (l'emploi du caractère $ est autorisé lorsque le nom du fichier ne change pas).

L'on transférera de la même manière les fichiers SYSTEM.MISCINFO, SYSTEM.BIOS, SYSTEM.INTERP et SYSTEM.Filer. Les choses deviennent déjà plus compliquées avec le SYSTEM.LIBRARY, ce fichier étant localisé sur la disquette PASCAL 3. PASCAL 3 doit être dans le drive # 5 lorsque le système la recherche. Il faut alors échanger à plusieurs reprises PASCAL 3 et PA4, sur le drive # 5, le système ne pouvant lire, puis écrire, qu'une partie à chaque fois. L'écran fournit des indications sur les changements de disquettes.

Le processus sera répété pour la disquette PA4MFP. La seule différence est le fichier SYSTEM.INTERP. Nous avons besoin, ici, de l'interpréteur à virgule flottante de la disquette UTIL 2. Le fichier UTIL 2 : FP.INTERP est transféré sur PA4MFP ; SYSTEM.INTERP sur PA5. SYSTEM.EDITOR se trouve sur PASCAL 1, SYSTEM.COMPILER sur PASCAL 2 (fichier assez long, nécessitant de nombreux échanges des disquettes sur le drive # 5), SYSTEM.LINKER sur PASCAL 3. Le système est maintenant prêt à fonctionner ; si l'on n'emploie que des nombres entiers (INTEGER), on mettra la disquette PA4 dans le lecteur # 4, et si l'on utilise aussi des nombres réels (REAL), il faut employer PA4MFP. PA5 est toujours placée dans le lecteur # 5.

Différences entre PASCAL UCSD et PASCAL standard

Déclaration CASE (1)

En PASCAL standard, le résultat de la déclaration CASE est indéfini lorsqu'aucune des conditions placées après CASE n'est réalisée. PASCAL UCSD, par contre, prévoit dans ce cas particulier que c'est la déclaration succédant à la déclaration CASE qui est exécutée.

COMMENTS

En PASCAL UCSD, tout ce qui est placé entre les symboles « (* » et « *) » ou « { » et « } » est considéré comme étant du commentaire ; une exception : lorsque le premier caractère est un « $ », auquel cas ce qui suit sera considéré comme étant un ordre de contrôle de compilation.

DISPOSE

L'instruction DISPOSE, définie en PASCAL standard, ne l'est pas en PASCAL UCSD.

EOF (F)

Voir UTIL:SETUP

FILES

Les fichiers du type interactif se comportent exactement comme les fichiers du type TEXT.

GOTO et EXIT

En PASCAL UCSD, toute déclaration GOTO se référant à une étiquette n'appartenant pas au même bloc est illicite.

EXIT(Q) signifie que la procédure en cours est terminée et que l'on exécute actuellement la procédure Q.

PACKED ARRAYS

Exemple : TABLEAU:PACKED ARRAY(0...9)OF CHAR ;

L'emploi de « PACKED » économise de la place mémoire, car chaque mot de seize bits contient deux caractères de huit bits (un caractère seulement si l'on n'emploie pas « PACKED »).

---

(1) N.d.t : « CASE statement » en anglais.

EN-TETE

PROGRAM DEMO (INPUT, OUTPUT) ; et

PROGRAM DEMO ;

sont, en fait, totalement équivalents en PASCAL UCSD puisque, dans tous les cas, le système déclare trois fichiers : INPUT, OUTPUT et KEYBOARD.

Les fonctions sur chaînes en PASCAL UCSD

LENGTH (STRING)

Donne le nombre (entier) de caractères de la chaîne. Exemple :

```
ASTRING: = 'EXEMPLE'
WRITELN(LENGTH(ASTRING)) ;
```

affiche : 8

POS(SUBSTRING, STRING)

Donne la position (nombre entier) d'une sous-chaîne à l'intérieur d'une chaîne. Exemple :

```
ASTRING: = 'CECI EST UN EXEMPLE' ;
BSTRING: = 'PLE' ;
```

affiche : 16

Remarque : lorsqu'une sous-chaîne ne se trouve pas dans la chaîne en question, la réponse est 0.

CONCAT(STRING)

Cette instruction concatène les chaînes les unes aux autres ; celles-ci seront séparées par une virgule.

```
Exemple :   ASTRING : = 'EXEMPLE' ;
            BSTRING : = 'CECI EST UN EXEMPLE' ;
            BSTRING : = CONCAT (ASTRING,'–', BSTRING) ;
            WRITELN (BSTRING) ;
affiche             EXEMPLE-CECI EST UN EXEMPLE
```

Copy (chaîne, position, nombre)

Définit une sous-chaîne dans une chaîne ; exemple :

```
            ASTRING : = 'TARTINE' ; BSTRING : = =COPY (AS-
            TRING,7,4) ;
            WRITELN (BSTRING) ;
affiche     TINE
```

Autre possibilité pour BSTRING :

```
BSTRING := COPY (ASTRING, POS('TI',ASTRING),4) ;
```

qui donne le même résultat.

Delete (chaîne, position, nombre)

Supprime un certain nombre (déterminé par l'utilisateur) de caractères à l'intérieur d'une chaîne.

```
Exemple :   ASTRING := 'CETTE CHAINE EST PRESQUE TROP
            LONGUE' ;
            DELETE (ASTRING, 19.5) ;
            WRITELN (ASTRING) ;
donne :     CETTE CHAINE EST TROP LONGUE
```

Insert (sous-chaîne, chaîne, position)

Insertion d'une sous-chaîne (inverse de DELETE).

```
Par exemple : ASTRING := 'INSERTION DE' ;
              BSTRING := 'DEMONSTRATION' ;
              INSERT (ASTRING, BSTRING,
```

Routines d'emploi fréquent en PASCAL UCSD

SIZEOF (VARIABLE) ;

Détermine la longueur, en octets, d'une variable.

LOG (NOMBRE) ;

Donne le logarithme de base 10 d'un nombre.

PWROFTEN (EXPOSANT) ;

Donne l'exposant en puissante de 10 (pour exposants de 0 à 37).

HALT ;

Peut être inséré dans tout programme ; provoque une erreur du temps d'exécution (run-time error) (# 14).

GOTOXY (XCOORD, YCOORD) ;

Envoie le curseur à l'emplacement déterminé par les coordonnées. Le coin supérieur gauche a les coordonnées (0,0), les autres positions variant suivant le terminal.

MEMAVAIL ;

Donne le nombre de cellules mémoires encore disponibles.

Exercice de réflexion :

Supposons que quelqu'un vous pose le problème suivant :

Vous disposez d'une somme de mille francs avec laquelle vous devez acheter trois sortes de vins différents, au minimum une bouteille de chaque, au total cent bouteilles, le prix total étant exactement de cent francs. Les prix des trois vins sont respectivement de 100, 30 et 5 F. Nous avons donc trois inconnues : les trois nombres de bouteilles, que nous nommerons X,Y et Z. Les données précédentes nous permettent d'établir deux équations, comme nous devrions le savoir depuis l'école ; ce sont :

$$X+Y+Z=100$$

puisque nous devons acheter 100 bouteilles en tout, et

$$100*X+30*Y+5*Z=1000$$

puisque le prix doit être de cent francs.

Nous ne pouvons pas déduire d'autres équations, à partir des données ci-dessus. Les systèmes à trois inconnues nécessitent cependant au moins trois équations ; nous ne pourrons donc résoudre notre système qu'en « tâtonnant ». A l'ère de l'ordinateur, ce procédé apparaît comme étant trop stupide et trop long ; l'on trouvera donc ci-dessous deux programmes apportant une solution au problème. Le premier est écrit en BASIC, le second en PASCAL. Bien que tous deux aient la même fonction, leur construction est très différente. Comme on le voit, le programme PASCAL se passe totalement de sauts, étant au contraire structuré en blocs délimités par des instructions BEGIN et END. Ces deux programmes contiennent chacun deux boucles imbriquées l'une dans l'autre, chacune comportant 99 passages.

Il est intéressant de comparer les temps de calcul des deux programmes : le programme BASIC, sur Superboard, UC 6502 à 1 MHz et interpréteur Microsoft, met 8 secondes à trouver la première (et unique) solution, et 199 secondes à parcourir toutes les possibilités. Les temps correspondants, pour le programme préalablement compilé, sur C4PMF, sont d'à peine deux secondes, et de 32 secondes !

```
10 REM PROGRAM ESSAI
20 FOR X=1 TO 98
30 FOR Y=1 TO 98
40 Z=100 -X-Y
50 IF 100*X+30*Y+5*Z=1000 THEN 90
```

```
60 NEXT X
70 NEXT Y
80 END
90 PRINT X;" BOUTEILLES A 100F"
100 PRINT Y;" BOUTEILLES A 30F"
110 PRINT Z;" BOUTEILLES A  5 F"
120 GOTO 80
```

```
PROGRAM ESSAI;

VAR Z:REAL;
    X,Y:INTEGER;

BEGIN
  X:=0;
  WHILE X < 99 DO
  BEGIN
    Y:=0;
    X:=X+1;
    WHILE Y < 99 DO
    BEGIN
      Y:=Y+1;
      Z:=100 -X-Y;
      IF 100*X+30*Y+5*Z=1000 THEN
      BEGIN
        WRITELN(X,' BOUTEILLES A 100F');
        WRITELN(Y,' BOUTEILLES A  30F');
        WRITELN(Z,' BOUTEILLES A   5F');
      END;
    END;
  END;
END
```

Stockage des programmes personnels

Après avoir tapé un programme à l'aide de l'éditeur, après que le fichier de travail ait été mis à jour et que la compilation ait été réalisée avec succès, la disquette du lecteur # 4 se trouve avec deux fichiers contenant chacun le programme : la version lisible

(pour l'utilisateur), sous le nom SYSTEM.WRK.TEXT, et la version compréhensible par la machine, sous le nom SYSTEM.WRK.CODE.

Si l'on désire maintenant travailler sur un autre programme, tout en conservant le programme actuel, l'on doit transférer ces deux fichiers sur une autre disquette, les nouvelles versions s'inscrivant par-dessus les anciens fichiers de travail.

Préparation de la disquette vierge :

La disquette vierge sera tout d'abord formatée (à l'aide du fichier DISKUTIL de la disquette UTIL 3) ; se référer au chapire « réalisation des fichiers » (ci-dessus), où l'on trouvera aussi la manière de donner un nom à la disquette et de lui déclarer un directory. Le nom pourrait être, par exemple, « MYDISK ».

Nous pouvons maintenant transférer le fichier de travail : l'on se place dans le fichier et l'on appuie sur T pour T(transfer.

Le système demande :

Transfer what file ?
Réponse : PA4+(ou PA4MFP, suivant la disquette placée dans le lecteur 4)
PA4:SYSTEM.WRK.TEXT (RETURN)

Le système cherche le fichier en question et demande, une fois qu'il l'a trouvé :

Transfer to where ?
Réponse : MYDISK:NOM.TEXT (RETURN)

Evidemment, la disquette MYDISK doit avoir été placée dans le lecteur # 5. Le fichier peut recevoir n'impote quel nom. L'on transférera de la même manière
PA4:SYSTEM.WRK.CODE sur
MYDISK:NOM.CODE.

Les instructions de l'éditeur de fichiers

G(et : cette instruction charge le fichier indiqué sur le fichier de travail. Si l'utilisateur n'indique que le nom du fichier, et que les fichiers texte et code existent, ces deux fichiers sont chargés, l'un sous le nom SYSTEM.WRK.TEXT, et l'autre sous le nom SYSTEM.WRK.CODE.

S(ave : permet de stocker le contenu actuel du fichier SYSTEM.WRK.TEXT sous un nom indiqué par l'utilisateur. Si, à ce fichier, correspond aussi une version en code, alors celle-ci sera stockée aussi, le fichier-texte le sera sous le nom TEXT et le fichier-code sous le nom CODE.

N(ew : efface le fichier de travail.

Q(uit : quitte l'éditeur de fichiers et rejoint le niveau de commande superficiel.

W(hat : donne le nom du fichier de travail et indique s'il a été sauvegardé ou non.

V(olumes : liste des périphériques actuellement « en ligne », avec leurs numéros.

L(dir : liste le directory de la disquette indiquée par l'utilisateur.

E(xtended List : liste une version étendue du directory.

C(hange : sert à modifier les noms des fichiers. Introduire d'abord le nom du fichier à modifier, puis le nouveau nom.

R(emove : sert à supprimer des noms de fichiers du directory ; ne doit pas être employé pour effacer SYSTEM.WRK.TEXT et SYSTEM.WRK.CODE, la commande à employer ici étant N(ew. L'ordinateur demande : Update directory ? Si l'on répond par Y, le nom est enlevé.

T(ransfer : sert à transférer des fichiers. Le système demande tout d'abord : Transfer what file ? et nous indiquerons le nom de la disquette et celui du fichier. Lorsque le système a trouvé ce dernier, il demande : Transfer to where ? Nous répondrons par le nom de la disquette et par le nouveau nom du fichier. Si on ne désire pas changer le nom du fichier, on peut taper un $ au lieu de répéter le nom.

M(ake : sert à inscrire un nom de fichier dans le directory.

Z(ero : cette instruction sert à aménager un directory. On l'emploie pour la mise en œuvre de nouvelles disquettes. Le directory consomme six blocs, et, lorsqu'on le copie, dix blocs de la disquette.

Commandes du niveau E(dit :

Le niveau E(dit sert à introduire de nouveaux programmes, ou à modifier les programmes existants.

J(mp pour JUMP affiche le texte suivant :
JUMP:B(eginning E(nd M(arker

Ici, B permet de placer le curseur au début du fichier, tandis que

E le déplace vers la fin de ce dernier. M permet de sauter sur un marqueur (l'ordinateur demande de quel marqueur il s'agit).

P(age : translate le curseur à fond vers le haut ou vers le bas.

I(nsert : est employé pour insérer (ou même tout simplement introduire) un texte. En l'absence d'un fichier de travail, I(nsert permet l'introduction d'un nouveau programme. Lorsque l'on désire insérer quelque chose dans un programme existant, il suffit de placer le curseur sur le caractère devant lequel l'on désire effectuer l'insertion, et d'appuyer sur I. Le texte, à partir du caractère en question, se déplace vers la fin de la ligne, ce qui permet de commencer l'insertion. Lorsque l'on a terminé, on appuie sur ETX(CTRL C).

D(elete : sert à effacer des caractères : on place le curseur sur le premier caractère à effacer, on appuie sur D, puis, aussi souvent que nécessaire, sur SPACE, jusqu'à ce que tous les caractères désirés soient effacés.

ETX(CTRL C) termine D(elete et remplit l'espace ainsi créé par translation de ce qui suit.

Z(ap : sert à effacer l'intégralité du texte, du début jusqu'à la position actuelle du curseur.

C(opy : sert à charger un fichier autre que le fichier de travail, et à l'éditer. L'ordinateur demande :

COPY : FROM WHAT FILE (MARKER, MARKER) ?

L'on indique alors le fichier désiré. Si le fichier comporte des marqueurs et que l'on ne désire pas le reprendre entièrement, il suffit d'indiquer le marqueur initial et le marqueur final de ce qui est à charger.

eX(change : cette instruction est employée pour échanger un caractère contre un autre, dans un texte. On quitte le mode EXCHANGE par la séquence ETX(CTRL C).

Les sept programmes suivants, appelés programmes de Bench Mark, sont construits pour pouvoir comparer les temps de calcul de différentes opérations, sur différents ordinateurs.

D'un autre côté, ces programmes sont aussi une très bonne démonstration de ce que peut être la simplicité et la clarté du langage de programmation PASCAL.

```
PROGRAM TRIANGLE;
VAR A,B,N:INTEGER;
BEGIN
  N:=7;
  FOR A:=1 TO N DO
  BEGIN
    WRITE(' ':N-A+1);
    FOR B:=1 TO 2*A-1 DO WRITE('*');
    WRITELN
  END;
END.
```

```
PROGRAM CODE;
VAR A:INTEGER;
    B:CHAR;
BEGIN
  WRITE('DECALAGE:');READLN(A);
  WHILE NOT EOF DO
  BEGIN
    READ(B);
    IF(B >='A') AND (B <='Z')
    THEN B:=CHR((ORD(B)+A+13) MOD 26+ORD('A'));
    WRITELN('=',B);
  END;
END.
```

```
PROGRAM EQLIN;
VAR A,B,C,D,E,F,X,Y,DET:REAL;
BEGIN
  WRITELN('RESOLUTION DU SYSTEME');
  WRITELN('AX+BY=C');
  WRITELN('DX+EY=F');
  WRITELN;
  WRITE('INDIQUER A,B,C  ');READLN(A,B,C);
  WRITE('INDIQUER D,E,F  ');READLN(D,E,F);
  WRITELN;
  DET:=A*E-B*D;
```

```
   IF DET=0 THEN WRITE('AUCUNE SOLUTION')
            ELSE BEGIN
            X:=(C*E-B*F)/DET;
            Y:=(A*F-C*D)/DET;
            WRITELN;
            WRITE('SOLUTION');
            WRITE('  X=',X:5:2);
            WRITE('  Y=',Y:5:2);
            END;
END.

PROGRAM BENCH1;
VAR K:INTEGER;
BEGIN
  WRITELN('DEBUT 1');
  FOR K:=1 TO 1000 DO
  BEGIN END;
  WRITELN('FIN');
END.


PROGRAM BENCH2;
VAR K:INTEGER;
BEGIN
  WRITELN('DEBUT 2');
  K:=0;
  WHILE K < 1000 DO
  BEGIN
    K:=K+1;
  END;
  WRITELN('FIN');
END.


PROGRAM BENCH3;
VAR K,A:INTEGER;
BEGIN
  WRITELN('DEBUT 3');
```

```
  K:=0;
  WHILE K < 1000 DO
  BEGIN
    K:=K+1;
    A:=K DIV K*K+K-K;
  END;
  WRITELN('FIN');
END.

PROGRAM BENCH4;
VAR K,A:INTEGER;
BEGIN
  WRITELN('DEBUT 4');
  K:=0;
  WHILE K < 1000 DO
  BEGIN
    K:=K+1;
    A:=K DIV 2*3+4-5;
  END;
  WRITELN('FIN');
END.

PROGRAM BENCH5;
VAR K,A:INTEGER;
PROCEDURE P;
BEGIN
END;
BEGIN
  WRITELN('DEBUT 5');
  K:=0;
  WHILE K<1000 DO
  BEGIN
    K:=K+1;
    P;
    A:=K DIV 2*3+4-5;
  END;
  WRITELN('FIN');
END.
```

```
PROGRAM BENCH6;
VAR K,A,L:INTEGER;
    M:ARRAY[1..5]OF INTEGER;
PROCEDURE P;
BEGIN
END;
BEGIN

  WRITELN('DEBUT 6');
  K:=0;

  WHILE K < 1000 DO
  BEGIN
    K:=K+1;
    P;
    A:=K DIV 2*3+4-5;
    FOR L:=1 TO 5 DO
    M[L]:=L;
  END;
  WRITELN('FIN');
END.

PROGRAM BENCH7;
VAR H,I,J,K:INTEGER;
    A:ARRAY[1..100]OF INTEGER;
BEGIN
  WRITELN('DEBUT 7');
  K:=0;J:=1;
  FOR H:=1 TO 1000 DO
  FOR I:=1 TO 100 DO
  J:=K;
  WRITELN('FIN');
END.

PROGRAM BENCH8;
VAR H,I,J,K:INTEGER;
    A:ARRAY[1..100]OF INTEGER;

BEGIN
```

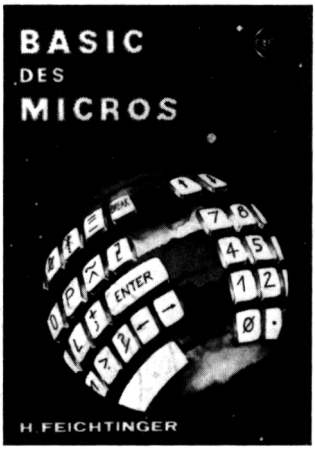
```
  WRITELN('DEBUT 8');
  K:=0;J:=1;
  FOR H:=1 TO 1000 DO
  FOR I:=1 TO 100 DO
  A[I]:=K;
  WRITELN('FIN');
END.
```

```
PROGRAM BENCH9;
VAR I:INTEGER;
BEGIN
  FOR I:=0 TO 1023 DO
  WRITE('.');
END.
```

H. FEICHTINGER

LE BASIC DES MICRO-ORDINATEURS

Par une comparaison pratique des différents MICROS travaillant en BASIC, cet ouvrage vous permettra d'apprécier les matériels les plus répandus.
Des glossaires de **vocabulaire** et une explication détaillée des **instructions** BASIC de chacun des appareils vous aideront à **perfectionner** votre programmation et à **adapter** aisément des programmes réalisés pour d'autres micros.



Principaux chapitres :

- **fonctionnement des micros**
- **différents modèles de micros**
- **termes et concepts à retenir**
- **instructions des divers BASIC**
- **écriture des programmes**
- **exemples de programmes en BASIC**

Un livre de 192 pages,
format 15 × 21 cm

Edité par :

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue - 75940 PARIS CEDEX 19

Collection Micro-Systèmes - ETSF **P. GUEULLE**

MAITRISEZ VOTRE ZX 81

Après vous avoir fait partager son apprentissage du Basic dans « Pilotez votre ZX 81 », Patrick Gueulle vous propose de découvrir la **programmation 16 K** et la **programmation en langage machine.**
L'assembleur Z 80 permet, grâce aux fonctions **PEEK, POKE** et **USR,** d'écrire des programmes extrêmement rapides, très peu encombrants et ouvrant la porte à des fonctions nouvelles telles que **les interfaces** auxquelles un chapitre entier est consacré.

Principaux chapitres :

- **Programmation avec le module 16 K**
- **Explorez la mémoire de la machine**
- **Les interfaces : introduction au langage machine**
- **Initiation au langage machine du ZX 81**
- **Du Basic au langage machine**
- **Jeu d'instructions machine du ZX 81.**

Un livre de 160 pages, format 15 × 21 cm

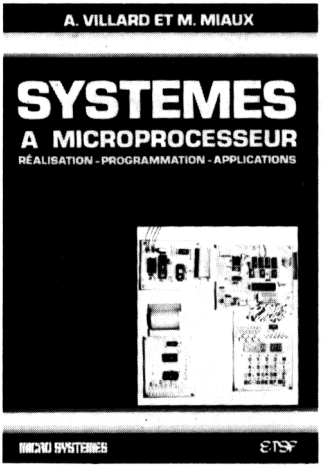
Edité par :

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue - 75940 PARIS CEDEX 19

Collection Micro-Systèmes-ETSF

A. VILLARD
et M. MIAUX

SYSTEMES A MICRO-PROCESSEUR :
réalisation, programmation, applications



Les auteurs, professeurs d'électronique, présentent la conception et la réalisation d'un système original permettant de mener à bien toute application réelle à microprocesseur. Vous pouvez étudier et mettre au point vos programmes grâce à un **moniteur.** Un **programmateur d'EPROM** permet leur transfert en mémoire morte et une **interface cassette** rend possible leur sauvegarde.

Un livre de 312 pages,
format 15 × 21 cm

Principaux chapitres :

- **Présentation de l'ouvrage et du microprocesseur utilisé**
- **Conception et réalisation des quatre maquettes du système**
- **Fontion et procédure d'utilisation du moniteur**
- **Affichage, scrutation et encodage du clavier**
- **Techniques de sous-programmes**
- **Le moniteur : description logicielle**
- **Le programmateur d'EPROM**
- **L'interface cassette**
- **Exemples d'applications.**

Edité par :

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue - 75940 PARIS CEDEX 19

Achevé d'imprimer
sur les presses
de l'Imprimerie Marcel Bon
70001 Vesoul
Dépôt légal : février 1983
N° éditeur : 373
N° imprimeur : 2664

DU BASIC AU PASCAL :
introduction au Pascal

Le Pascal, par sa **construction logique** est d'un **apprentissage facile** et, de plus, il incite le programmeur à écrire des **programmes clairs.**
Le présent ouvrage s'efforce d'une part de permettre l'**accès au Pascal** et, d'autre part, à l'intention de tous ceux qui, jusqu'à présent, n'utilisèrent que le Basic, de faciliter la **reconversion au Pascal,** les premiers programmes étant accompagnés de leur équivalent en Basic.

Principaux chapitres :

- Pourquoi le Pascal ?
- Les éléments de base du Pascal
- La section de déclaration
- Les boucles et les imbrications
- Données et ensembles de type particulier
- Procédures et fonctions
- Structures de données dynamiques
- Collection de différents programmes
- Comment élaborer un programme en Pascal.
- Intégration de programmes machine aux programmes en Pascal

ISBN 2-85535-023-9

TARIF CODE L

ETSF

E. FLOEGEL

DU BASIC AU PASCAL

4